VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Mobilní aplikace pro usnadnění cestování uživatelům s mentálním postižením

Mobile Applications to Facilitate Travelling for Users with Mental Disabilities

2016 Josef Raška

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Josef Raška**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Mobilní aplikace pro usnadnění cestování uživatelům s mentálním postižením
Mobile Applications to Facilitate Travelling for Users with Mental Disabilities

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem diplomové práce je vytvoření mobilní aplikace pro mobilní platformu Android, která se bude snažit o to, aby za pomocí technologií GPS a NFC uživatelům s mentálním postižením usnadnila každodenní cestování a s tím spojené činnosti. V rámci aplikace bude uživateli umožněno za pomocí asistenta nahrát několik obvyklých tras, při jejichž následném novém průchodu bude aplikace pomáhat. K dané cestě bude možno v aplikaci nahrát komentáře, fotky, hlasové záznamy nebo video. Asistent může dále nastavit upozornění pro uživatele, kdy má na cestu vyrazit, co si má vzít s sebou atd. Součástí aplikace budou také funkce jako upozornění na to, že během cestování může dojít baterie, statistiky průběhu cestování a kontaktování asistenta v případě potřeby s informací o poloze uživatele. Samotná navigace pak bude dostupná v několika úrovních, aby se uživatel cesty postupně sám učil a nebyl zcela závislý na aplikaci. Pro usnadnění vstupu a zrychlení vyhledávání konkrétní trasy v seznamu nahraných tras bude použita technologie NFC.

Jednotlivé body práce jsou:
1. Analýza a návrh mobilní aplikace.
2. Implementace jednotlivých funkcionalit mobilní aplikace.
3. Implementace web rozhraní pro asistenty.
4. Testování aplikace v reálném prostředí.
5. Vyhodnocení testování a návrh na možných rozšíření aplikace.

Seznam doporučené odborné literatury:

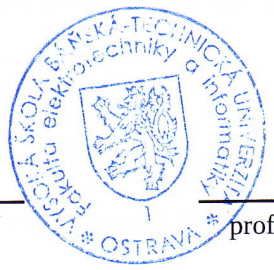
[1] Grant Allen: Android 4, překlad Mužík Jakub, Computer Press, 2013, http://neoluxor.cz/odborne-knihy/android-4--194845/
[2] Android Developers, http://developer.android.com/develop/index.html
[3] http://www.nfctech.cz

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Martinovič, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 29.04.2016



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2016 Josef Raška

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 29. dubna 2016 ..Josef Raška.....

Rád bych poděkoval mému vedoucímu Ing. Janu Martinovičovi, Ph.D. za pomoc, připomínky a vedení této práce.

Dále děkuji paní Mgr. Barboře Uhlířové ze společnosti SPMP za konzultace a pomoc při formulaci zadání.

Velmi děkuji mým přátelům Ing. Janu Poláčkovi, Bc. Jiřímu Mikovi, Bc. Simoně Raškové a mnohým dalším, kteří mou aplikaci detailně otestovali a poskytli mi cennou zpětnou vazbu a názory, které aplikaci vylepšily.

Také děkuji panu Lubomíru Teichmannovi a Tamaře Iskrytskaye ze společnosti Zdravotní a sociální služby s.r.o., kteří aplikaci otestovali s klienty a poskytli důležité nápady pro její další úpravy a vývoj.

Abstrakt

Cílem práce je analyzovat problémy, se kterými se setkávají lidé s mentálním postižením při jejich každodenním cestování, vymyslet a implementovat za pomoci dostupných nástrojů mobilní aplikaci na platformu Android, řešící tyto úkony a následně ji publikovat a otestovat. V rámci práce byla navržena a vyvinuta Android aplikace pro asistenci cestování, která by měla lidem s mentálním postižením usnadnit jejich obvyklé cesty, poskytnout jim prostředek pro jejich uchovávání a urychlit komunikaci s jejich asistentem. Aplikace využívá pro asistenci systému GPS, pro efektivní implementaci množství aktuálních knihoven, postupů a pro zjednodušené zrychlené zadávání vstupu do aplikace byla využita technologie NFC. Výsledná aplikace byla publikována do obchodu Google Play a následně otestována v městském prostředí, kde uživatelům může přinést požadovanou pomoc. Implementace má volně přístupné zdrojové kódy a ty tak mohou být zdrojem pro její další rozvoj a širšímu využití.

Klíčová slova: Android, vývoj, asistence cestování, mobilní aplikace, mentální postižení, aplikační návrh, GPS, NFC, Google Play, GitHub

Abstract

The aim of the thesis is to analyze problems, which the people with mental disabilities deal with during every day traveling, invent and implement mobile application on Android platform with available tools, which will solve these issues, then publish and test it. Android application for travel assistance was designed and developed as part of this thesis, which should help the people with mental disabilities make their usual travels easier, provide them tool for saving them and speed up communication with their assistant. Application uses GPS system for assistace, lot of actual libraries and methodologies for effective implementation and NFC technology was used for faster and easier input actions. Application was published to Google Play store and then tested in urban environment, where the users can benefit from it. Implementation is completely open source and the code could be source for its next development and broader usage.

Key Words: Android, development, travel asistance, mobile application, mental disabilities, application design, GPS, NFC, Google Play, GitHub

Obsah

Seznam použitých zkratek a symbolů

SDK	–	Software development kit (*Soubor nástrojů pro vývoj*)
GPS	–	Global Positioning System (*Globální systém určování polohy*)
NFC	–	Near field communication (*Technologie pro přenos dat*)
API	–	Application programming interface (*Rozhraní pro programování aplikací*)
UML	–	Unified Modeling Language (*Grafický jazyk v softwarovém inženýrství*)
SD karta	–	Paměťová karta technologie Secure Digital
ORM	–	Objektově relační mapování
ZIP	–	Souborový formát pro kompresi a archivaci dat
HTTP	–	Hypertext Transfer Protocol (*Protokol pro přenos webových dokumentů*)
REST	–	Representational State Transfer (*Architektura rozhraní HTTP komunikace*)
SOLID	–	Single responsibility, open-closed, Liskov substitution principle, interface segregation and dependency inversion (*Jedna odpovědnost, otevřenost k rozšíření, uzavřenost ke změnám, princip Liskové substituce, oddělení rozhraní a obrácení závislostí*)
IoC	–	Inversion of control (*Obrácení závislostí*)
MVC	–	Model View Controller (*Typ architektury aplikací*)
MVP	–	Model View Presenter (*Typ architektury aplikací*)
JAR	–	Java Archive (*Archiv souborů jazyka Java*)

Seznam obrázků

1 Úvod

Problém cestování a orientace v reálném čase je v dnešním světě občas složitým problémem pro nás všechny, pro lidi s mentálním postižením pak problémem mnohonásobným. Cesta do zaměstnání, do banky nebo jen do obchodu může být pro člověka s mentálním postižením velmi těžký úkol, jelikož často musí využít hromadné dopravy, přecházet složité křižovatky a podobně. Lidí s mentálním postižením je momentálně v České republice více než 100 tisíc [10] a jedná se tedy o velice významnou skupinu, která si zaslouží zvláštní pozornost. Tito lidé pak při svém cestování často potřebují pomoc asistenta, kterých je však pro takto početnou skupinu nedostatek. Dnes jsou již chytré mobilní telefony poměrně snadno dostupnou věcí, spoustu lidí s mentálním postižením tyto telefony více než zdatně ovládá a nabízí se tedy možnost pokusit se prostřednictvím této techniky těmto lidem pomoci. V tomto případě mobilními zařízeními se systémem Android, který je v současné době nejrozšířenějším a také nejdostupnějším mobilním operačním systémem.

Jako téma diplomové práce bylo proto zvoleno vytvoření mobilní aplikace pro mentálně postižené, která se pokusí usnadnit jim cestování a urychlit komunikaci s asistentem. V práci se nejprve seznámíme s platformou Android a všemi nástroji a knihovnami využitými při vývoji aplikace. Pokračovat budeme procesem vytváření zadání aplikace na základě podrobné diskuze k dané problematice s povolanými osobami, dále si představíme funkční a grafický návrh aplikace. Následovat pak bude popis implementace za použití představených nástrojů a knihoven a následně ukážeme, jak je možné aplikaci propojit se službami třetích stran pro rozšíření její funkcionality, jak můžeme aplikaci publikovat a zpřístupnit ji snadno uživatelům. Závěrem budou popsány změny a úpravy provedené po obdržení zpětné vazby testovacích uživatelů a případné možnosti dalšího vývoje tohoto typu aplikace.

2 Platforma Android

Systém Android byl poprvé vypuštěn do světa v roce 2008 v útrobách zařízení T-Mobile G1 [14] a díky své dostupnosti a otevřenému kódu se rychle rozšířil a v současné době je to zcela jasně nejvíce využívaný mobilní operační systém, který se stále bouřlivě vyvíjí. Jeho otevřenost je zároveň pro vývojáře i jedním z jeho problémů, neboť výrobci si systém rádi upravují, v důsledku čehož se může aplikace na některých zařízeních chovat neočekávaně a často se lze setkat se situací, že určitá funkce aplikace nefunguje správně právě na jednom konkrétním typu zařízení.

Díky své rozšířenosti a dostupnosti se platforma Android nabízí jako dobrá možnost pro implementaci aplikace na podporu cestování, neboť bude pro klienty snadno dostupná na zařízeních všech cenových kategorií, kam si je mohou stáhnout přes obchod Google Play, kde bude aplikace publikována. Systém nabízí podporu pro velké množství různých senzorů, které dnešní mobilní zařízení obsahují a tak je čtení dat z akcelerometru nebo zaznamenávání GPS řešeno pouze voláním příslušných metod rozhraní frameworku. Během doby existence systému se objevilo také mnoho rozšiřujících knihoven pro zjednodušení práce programátora umožňujících vyvíjet aplikace efektivněji. Naše aplikace se za pomoci těchto nástrojů pokusí zjednodušit mentálně postiženým klientům cestování po svých obvyklých cestách. V této kapitole se seznámíme se základním programovým rozhraním systému, představíme si použité vývojářské nástroje a také využité knihovny.

2.1 Základní komponenty programového rozhraní

Systém Android poskytuje vývojáři několik základních komponent, pomocí nichž by měl skládat svou aplikaci. Pochopení účelu těchto komponent je důležité pro orientaci v implementaci a proto si nejdůležitější námi používané komponenty blíže popíšeme.

2.1.1 Komponenta typu Záměr (`Intent`)

Platforma Android byla vymýšlena tak, aby vývojář místo explicitního spouštění konkrétní aplikace pro danou činnost raději popsal záměr, co by chtěl udělat a systém sám vyhodnotí, která aplikace se pro tento záměr nejlépe hodí. Touto cestou se systém snaží vytvořit prostředí mnoha aplikací, které spolu navzájem komunikují, aniž by jedna o druhé přímo věděly a pomocí záměrů tak snižuje vazby mezi komponentami. Záměr může mít textově popsanou obecnou akci, blíže specifikovanou kategorii, typ, mnoho dalších parametrů až po konkrétní jméno komponenty, která má záměr obdržet. Pomocí těchto nastavení systém vyhodnotí nejlepší komponentu, která má pro uživatele záměr vykonat. Při vyhodnocování záměrů systém využívá pro správný výběr tzv. filtry záměrů (`IntentFilter`).

Pokud chceme například odeslat email, nespouštíme emailovou aplikaci, ale vytvoříme záměr odeslat email s příslušnými parametry a tento záměr předáme systému, který ze seznamu nainstalovaných aplikací vybere ty, které dokážou náš email odeslat a v případě více výsledků nechá

uživatele vybrat, která z aplikací má záměr obdržet. Parametry záměru by v tomto případě byly předmět, adresát emailu a tak dále.

Naopak pokud chceme v aplikaci spustit některou z našich aktivit, dáváme aplikaci pomocí třídy aktivity přesné jméno komponenty, kterou má spustit, nicméně samotnou aktivitu s jejím životním cyklem vytvoří systém, nikoliv vývojář sám.

Pomocí záměru se předávají všemožné informace a může kromě zmíněných případů sloužit k vyvolání vzdálené akce, informace o systémové události nebo pouze pro přenos dat, které musí být vždy serializovatelné, neboť je tím umožněno systému záměr uchovat i v případě ukončení spouštěcího procesu a předávání záměrů mezi procesy.

2.1.2 AndroidManifest.xml

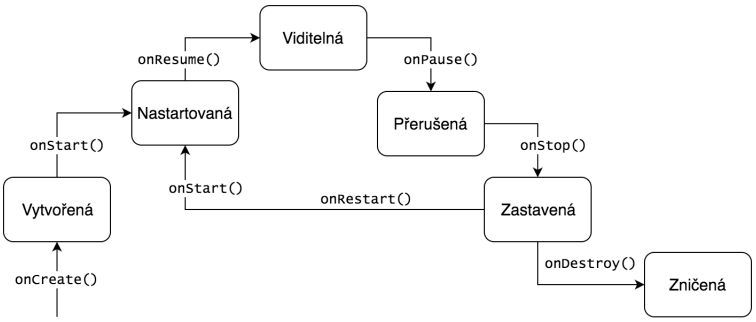
Jedná se v o výpis všech veřejných informací aplikace ve formátu XML, které systém a také uživatel potřebují o aplikaci vědět. Android manifest obsahuje identifikátor aplikace, deklaruje všechny požadovaná povolení, podporované velikosti obrazovek zařízení nebo vyžadované funkce jako například přítomnost fotoaparátu. Dále deklaruje podporované verze systému a v neposlední řadě také všechny hlavní komponenty aplikace, jako například aktivity a služby a tím definuje všechny vstupní body aplikace a podmínky jejich spuštění pomocí záměrů. Při publikování aplikace parsuje Google Play tento soubor pro zobrazení informací pro záznam v obchodě a podobně instalační služba systému Android pro zobrazení ikony v seznamu aplikací a také vytvoření uživatele podle identifikátoru aplikace a přiřazení přístupových práv pro konkrétní deklarovaná povolení.

2.1.3 Aktivita

Aktivita je komponenta aplikace, která obsahuje uživatelské rozhraní viditelné pro uživatele. Každá používaná aktivita aplikace musí být definována v Android manifestu se svými podmínkami pro své spuštění, neboli filtry záměrů (`IntentFilter`) a tím deklarovat vstupní místa do aplikace. Nejčastěji je v aplikaci jedna aktivita figurující jako spouštěč (akce záměru `MAIN`, kategorie `LAUNCHER`), která se při instalaci vyhledá a vytvoří se pro ni spouštěcí ikona v seznamu nainstalovaných aplikací. Další aktivity pak mohou mít explicitně definované vlastní filtry záměrů, případně nedefinovat žádný a tedy mít implicitně definovaný filtr na jméno třídy aktivity v kontextu balíčku aplikace.

Třída aktivity, dědící ze systémové třídy `Activity`, poskytuje vývojáři API se svým životním cyklem, skrz který by mělo být reagováno na to, v jakém stavu se uživatelské rozhraní právě nachází. Pro akce při vytváření aktivity je používaná metoda `onCreate`, která se volá pro každou instanci aktivity právě jednou a mělo by se v ní připravit uživatelské rozhraní, získat příslušné služby a podobně. Aktivita je zobrazena uživateli po zavolání metody `onResume` a v této chvíli může uživatel interagovat s uživatelským rozhraním. Pokud uživatel přechází do jiné aktivity, přepíná aplikace, zamyká obrazovka nebo jiným způsobem opouští uživatelské rozhraní, volá se metoda `onPause`. Pokud se zcela ukončuje používání konkrétní aktivity, volá se metoda

`onDestroy`, kde by se měly uvolnit všechny zdroje. Celý životní cyklus aktivity můžeme vidět na obrázku 1 a jedná se často o první diagram, který vývojář pro platformu Android vidí. Velmi častým problémem při vývoji je držení dat a podobných věci přímo v aktivitách, neboť ta může být kdykoliv zničena systémem, například pokud uživatel pouze otočí obrazovku a vyžaduje se v té chvíli nové uživatelské rozhraní. Pokud v takové chvíli někdo drží stále na aktivitu referenci, zničená instance aktivity není uvolněna z paměti a vzniká únik paměti. Tento problém se stal ve světě vývojářů Androidu tak častý, že přímo pro jeho prevenci byla vytvořena knihovna LeakCanary[1].



Obrázek 1: Životní cyklus aktivity

2.1.4 Fragmenty

Fragmenty byly představeny v Androidu verze 3 a zpočátku způsobily mnoho otázek a jejich správném využívání. V současnosti se doporučuje využít je hlavně pro dynamické zobrazování uživatelského rozhraní na zařízeních s různými obrazovkami [4] a pro logiku patřící pouze uživatelskému rozhraní. Fragmenty mají s aktivitou svázaný stejný životní cyklus a všechny metody tohoto cyklu jsou provolávány do fragmentu. Jedna instance fragmentu může být předávána mezi instancemi více aktivit a fragment tak může sloužit jako ideální komponenta v případě, kde potřebujeme mít logiku svázanou s životním cyklem aktivity, chceme však současně jistá data předat bezpečně mezi instancemi rotující aktivity, případně pokud chceme určitou část uživatelského rozhraní udržet i při změně instance aktivity. Toto v aplikaci používáme například u map, kde pomocí držení mapy ve fragmentu předcházíme novému načtení mapy při rotaci a tudíž nežádoucímu probliknutí.

2.1.5 Služby

Služby jsou určeny pro operace, které nejsou viditelné uživateli, případně asynchronní operace, které se mají provést bez vazby na uživatelské rozhraní. Služby mají také svůj životní cyklus, nicméně odlišný a jednodušší, než aktivity. Instance služeb, dědících ze systémové třídy `Service`, opět vytváří systém a spravuje je po dobu jejich existence, přičemž vývojář spouští služby záměry a pomocí nich jim předává také například data ke zpracování. Služby se stejně jako aktivity

[1] https://github.com/square/leakcanary (10.4.2016)

musí registrovat v Android manifestu a mohou být součástí veřejného rozhraní pro jiné aplikace. Pomocí deklarace v manifestu můžeme také definovat nový proces pro službu a ještě více tak oddělit její spuštění od naší aplikace.

2.1.6 Aplikace

Při spuštění aplikace se vytvoří právě jeden aplikační objekt, kde může vývojář inicializovat komponenty potřebné v celé aplikaci. Aplikace se opět deklaruje v manifestu jménem své třídy, která dědí ze systémové třídy `Application`.

2.2 Nástroje použité pro vývoj

Android je v současné době nejpoužívanější mobilní operační systém a programuje pro něj obrovské množství vývojářů, kteří pro stále vyšší nároky na aplikace potřebují pracovat efektivně a Android samotný a spoustu dalších společností pro ně poskytuje celou řadu nástrojů, které jim mají usnadnit práci a zefektivnit vývoj.

2.2.1 Vývojové prostředí

Přesto, že první verze Android vyšla již před 10 lety, získal Android své vlastní vývojové prostředí, Android Studio[2], až v roce 2013 a na verzi 1.0 si vývojáři počkali až do konce roku 2014. Dříve byl vývoj řešen pluginy ve známých vývojových prostředích jako Eclipse[3], NetBeans[4] a později IntelliJ IDEA[5] od JetBrains. Právě poslední jmenované se časem stalo jedničkou a díky tomu, že existuje open source verze, použily se zdrojové soubory IntelliJ jako základ pro Android Studio a dodnes se změny pro IntelliJ dostávají postupně do Android Studia a naopak, takže se vzájemně stále obohacují.

S příchodem vlastního vývojového prostředí se vývoj značně zjednodušil a tým Android Studia neustále vylepšuje integraci s SDK, nástroji pro sestavení, neboli build projektu a pluginy, kterých je celá řada a ten hlavní je Android Gradle Plugin nad open source build systémem Gradle[6]. Ten umožňuje definici sestavení v Groovy skriptech s možností využití Javy, pomocí něhož se celý projekt sestavuje a vytváří se aplikace se všemi dílčími kroky, které jsou pro to potřeba. Pomocí Gradle pluginu se do procesu sestavení tyto kroky vloží, umožní se jejich konfigurace pomocí Gradle skriptů a Android Studio slouží pouze jako obálka nad běžícím Gradle sestavením. Vývojové prostředí řeší pouze asistenci vývojáři, napovídání, analýzu kódu a celkově vývojářovo pohodlí, neřeší však samotné sestavení, které přenechává Gradlu a spuštění sestavení z příkazové řádky je naprosto ekvivalentní sestavení ve vývojovém prostředí, což znamená velkou výhodu při sestavování aplikace v cizích prostředích, například build serveru. Gradle také

[2] http://developer.android.com/sdk/index.html (24.4.2016)
[3] https://eclipse.org/ (24.4.2016)
[4] https://netbeans.org/ (24.4.2016)
[5] https://www.jetbrains.com/idea/ (24.4.2016)
[6] http://gradle.org/ (24.4.2016)

poskytuje velice užitečný mechanismus jménem Gradle Wrapper, což je spustitelný Java JAR archiv, který je spolu s několika skripty uložen spolu s projektem a při spouštění sestavení přes tyto skripty stáhne v rámci sestavení nakonfigurovanou distribuci Gradlu na spouštějící stanici. Při distribuci kódu tedy stačí pouze spustit skript a vývojář se nemusí starat o jakoukoliv komplikovanou instalaci build systému.

2.2.2 Verzování kódu

Jako systém verzování byl použit Git, který je jeden z nejrozšířenějších verzovacích systémů a služba GitHub[7], která podporuje zdarma hostování open source repozitářů s nabídkou mnoha služeb okolo, mimo jiné snadnou integraci se službou Travis CI, která byla použita pro průběžnou integraci projektu. Díky službě GitHub je zajištěno zálohování kódu během vývoje a díky Gitu je vyřešeno verzování se všemi výhodami, které to s sebou přináší, jako možnost vrátit se ke konkrétní verzi projektu, porovnávat a sledovat změny, případně vyvíjet několik funkcionalit nezávisle na sobě. Dále bylo pomocí těchto nástrojů řešeno uzavírání verzí publikováno na Google Play, kdy byl pro každou verzi vytvořen release, takže v případě, že se objeví problém u uživatele, jsme vždy schopní snadno dostat kód do stavu, ve kterém je u uživatele a snáze tak najít a vyřešit problém.

2.2.3 Průběžná integrace

Díky hostování projektu na GitHubu bylo možné snadné využití službu Travis CI[8], která se zaregistruje pro repozitář a při každé změně spustí sestavení celého projektu v uzavřeném prostředí, díky čemuž dokážeme okamžitě odhalit jakékoliv problémy, které se na naší stanici nemusí projevit, nicméně mimo naše prostředí ano. Služba stáhne veškerý kód z repozitáře, nakonfiguruje uzavřený virtuální kontejner, nainstaluje Javu, Android SDK, případně další komponenty a spustí sestavení projektu, v rámci něhož se spouští i unit testy. Služba se konfiguruje pomocí soubor `travis.yml` v kořenovém adresáři repozitáře, v němž je specifikováno, že se projekt má sestavit jako Android projekt, jaký software se do prostředí má před každým během nainstalovat a co se má stát v případě chyby. Travis CI po stažení repozitáře tento soubor vyhledá a na základě jeho obsahu provede požadované akce.

2.2.4 Statická analýza kódu

Pro analýzu kódu a odhalování potencionálních chyb, problémů a nepoužívaných souborů je používán nástroj **Lint**, což je nástroj, který v projektu kontroluje dodržení sady poskytnutých pravidel a který je již součástí sestavení procesu Androidu. Během sestavení se vytváří Lint Report se všemi varováními a chybami, které byly v projektu nalezeny. Nástroj je konfigurovatelný pomocí Gradle skriptů projektu, případně souborem `lint.xml`, pomocí něhož lze některé chyby

[7] Stránku projektu na GitHubu lze nalézt na https://github.com/jraska/Diploma-Thesis (29.4.2016).

[8] Službu a historii sestavení pro aplikaci Asistence cestování si můžeme prohlédnout na https://travis-ci.org/jraska/Diploma-Thesis (10.4.2016).

vynechat, případně změnit jejich prioritu například z chyby na varování. Lint dokáže odhalit spoustu chyb jako třeba zapomenutí potvrzení při editaci nastavení uživatele, neuzavření databázového kurzoru, přítomnost nepoužitých obrázků v projektu a podobně. Jeho další velkou výhodou je rozšiřitelnost, kde si každý může napsat vlastní Lint pravidla pro to, na co chce být upozorněn a v případě knihoven začne Android Studio jejich pravidla automaticky používat a zvýrazňovat problémy uživateli přímo v editoru. Příkladem může být knihovna používaná pro logování Timber, která pomocí Lint pravidel kontroluje, zda je použita správně a dokáže tak předejít chybám za běhu.

Dalším použitým nástrojem je služba **Codacy**[9], což je ve své podstatě automatizována analýza kódu vyhledávající potenciální chyby a problémy, která se spustí po každé změně v kódu díky opět snadné integraci s GitHubem. Služba odhaluje problémy jako nepoužívaný kód, příliš složité třídy nebo metody, známé programátorské chyby či bezpečnostní rizika. Po analýze projekt oznámkuje podle množství nalezených problémů v poměru k množství analyzovaného kódu.[10]

2.3 Použité knihovny

Jak již bylo zmíněno, Android používá velké množství vývojářů a pro jejich potřebu vzniklo velké množství z drtivé většiny open source knihoven, tedy knihoven s volně přístupným kódem, které se snaží vyřešit a zjednodušit společný problém mnoha vývojářů a usnadnit programování pro Android. Knihovny jsou díky tomu, že si může napsat knihovnu prakticky kdokoliv různých kvalit a je proto nutné si každou napřed prohlédnout před jejím použitím, nicméně díky jejich otevřenému kódu a hodnocením dalších uživatelů lze již nyní poměrně snadno najít spolehlivou knihovnu na téměř každý typický problém, který při vývoji aplikací nastává a pročtením jejího kódu se ujistit, případně inspirovat, jak dělat danou věc správně.

U některých knihoven je také podrobně zdůvodněno, proč jsou používány, neboť mnohé řeší velice časté problémy vývoje, příčiny problematického kódu a je na nich později založena architektura aplikace a metodika vývoje.

2.3.1 Android Support

Jedná se o základní knihovnu poskytovanou jako součást SDK, která řeší problémy kompatibility aplikace mezi různými verzemi a sjednocuje jejich vzhled. Poskytuje také množství rozšiřujících komponent uživatelského rozhraní, pomocné třídy například pro notifikace, média a spoustu dalších. V současné době by měla být součástí každé aplikace a při vytváření nového projektu je k projektu již automaticky přidána.

Dokumentaci knihovny můžeme najít na http://developer.android.com/tools/support-library/index.html (10.4.2016).

[9] https://www.codacy.com/ (10.4.2016)

[10] Aktuální analýzu projektu si můžeme prohlédnout na https://www.codacy.com/app/josef-raska/Diploma-Thesis/dashboard (29.4.2016).

2.3.2 RxAndroid

Jedná se o Android rozšíření známé knihovny RxJava, která implementuje reaktivní rozšíření mnohých programovacích jazyků sdružené pod projekt ReactiveX (http://reactivex.io/, 10.4.2016). Knihovna k původní Java implementaci pouze přidává vlastnosti specifické pro platformu Android a to konkrétně řešení spouštění metod v hlavním vlákně uživatelského rozhraní. Knihovna je založena na návrhovém vzoru Pozorovatel (Observer [3]) a její záměr je implementovat reakce na události bez vědomí, kdy přesně událost nastane, ale s vědomím, že v tuto chvíli bude registrovaný pozorovatel na tuto událost upozorněn a zareaguje na ni. Knihovna nabízí spoustu metod, jak na události reagovat, shlukovat je do skupin, mapovat objekty výsledku na jiné, řeší příliš mnoho požadavků v jednom čase a spoustu dalších vlastností. Jedná se o poměrně mocný nástroj, který nicméně vyžaduje nemálo času na nastudování, pochopení a v případě Android platformy platí, že s velkou sílou přichází velká zodpovědnost, jelikož při registrace pozorovatele se často v rámci registrace drží reference na aktivitu, což může v případě například rotace způsobit únik paměti, případně i pád aplikace v důsledku pokusu o aktualizaci uživatelského rozhraní, které v té době už nemusí být dostupné.

RxAndroid navíc řeší další, pro Android velmi významný problém a tím je výběr vlákna, ve kterém se operace spouští. Jako hlavní vlákno totiž slouží vlákno uživatelského rozhraní, které by nemělo provádět žádné, byť jen trochu náročné operace, neboť by se vše mělo stihnout během 16ms [19]. V aplikacích i knihovnách toto vede k obrovskému množství asynchronních zpětných volání, které mají různá jména, každá knihovna si je řeší po svém, není nad nimi kontrola a výsledkem je nepřehledný asynchronní kód, ve kterém se lze jen složitě orientovat. Díky RxAndroid a jejich implementaci plánovačů lze rozhodnutí, na jakých vláknech bude operace spuštěna a na kterém vlákně bude doručen výsledek nechat na klientovi třídy a tím pádem například řetězit synchronní volání, pokud jsme již v asynchronním vlákně a zejména zajistit, že výsledek obdržíme ve vlákně uživatelského rozhraní. Typická ukázka využití této vlastnosti lze vidět na ukázce kódu 1, kde se získá seznam cest, operace pro jeho získání, které jsou časově náročné se spustí v jiném vlákně a výsledek bude doručen na hlavní vlákno, kde spustí poskytnutou metodu.

```
    ...
    routesRepository.selectAll()
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(this::setRoutes);
}

void setRoutes(List<RouteData> routes) {...}
```

Výpis 1: Reakce na událost v UI vlákně pomocí RxAndorid

Výhody reaktivního programování můžeme ocenit i při testování, které je typicky s asynchronním rozhraním testované třídy problematické, nicméně díky RxAndroid volání `toBlocking()` vynutíme synchronní spuštění požadované operace, což nám výrazně zjednoduší testování.

Knihovnu můžeme nalézt na https://github.com/ReactiveX/RxAndroid (10.4.2016).

2.3.3 Retrolambda

Lambda výrazy jsou mocným nástrojem a příjemnou pomůckou každého jazyka, který je podporuje a redukují často dlouhé bloky kódu do kratších, přehlednější funkcí. Jelikož Android kvůli zpětné kompatibilitě podporuje pouze Javu 7[11] a lambda výrazy byly do Javy přidány až ve verzi 8, nemůže tyto funkce vývojář využívat, což vede často k velkému množství anonymních tříd pro zpětné volání mnoha metod, jako typicky klepnutí, vrácení načtených asynchronních dat nebo při registraci posluchače u výše zmíněné knihovny RxAndroid. Tyto anonymní třídy značně ztěžují čitelnost kódu a velmi často vedou také k únikům paměti, neboť anonymní třída drží implicitně referenci na instanci třídy, ve které je vytvářena, což je často aktivita nebo jiný objekt, který by měl být z paměti jinak odstraněn.

Retrolambda umožňuje lambda funkce využít i v rámci kódu podporujícího pouze Javu 7, čímž Android vývojáři umožňuje zbavit se anonymních tříd a využít potenciálu lambda funkcí ve své aplikaci při zachování zpětné kompatibility s bytekódem Javy 7. Pro její využití je třeba nastavit kompilaci projektu na verzi Javy 8 a do Gradle skriptů přidat plugin knihovny. Použité lambda výrazy se během Java 8 kompilace přeloží do bytekódu, vyžadující její běhové prostředí, nicméně tento nástroj transformuje bytekód do instrukcí, které jsou dostupné v běhovém prostředí Javy 7 a pokud je to možné, provede další optimalizace pro ušetření vytváření instance objektu pro každé volání.

Ukázku odstranění anonymní třídy můžeme vidět na ukázce kódu 2.

```
saveButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
          save();
        }
      });

          ↓                ↓

saveButton.setOnClickListener(v -> save());
```

Výpis 2: Odstranění kódu anonymní třídy pomocí knihovny Retrolambda

Knihovnu můžeme nalézt na https://github.com/orfjackal/retrolambda (10.4.2016).

[11] Vyjma `try-with-resources`, které je podporováno až od verze Androidu 5.0.

2.3.4 Dagger 2 a vkládání závislostí

Poměrně nová knihovna spravovaná Googlem a uvolněná poprvé v srpnu 2015, která pro nás řeší tzv. vkládání závislostí (anglicky Dependecy Injection) do objektů, které tyto závislosti chtějí používat. Knihovna je použita za účelem zkvalitnění kódu a zlepšení testovatelnosti, jelikož je v celém projektu následována filozofie, že třídy by měly objekty buď vytvářet a nebo používat. To v podstatě znamená, že se snažíme omezit situace, kdy si třída v rámci své metody vytváří vlastní objekty, které potřebuje k vykonání dané operace a zejména se snaží úplně eliminovat situace, kdy si třída své závislosti sama začne vybírat z nějakého kontejneru, statických proměnných a podobně. Jako špatný přístup je v tomto případě brán i často používán návrhový vzor Jedináček, jelikož se jedná v podstatě o sdílený globální stav. Všechny objekty by měly své závislosti obdržet v konstruktoru a neměly by se vyskytovat žádné metody typu inicializuj objekt, nastav proměnnou, bez které objekt nemůže fungovat a podobně.

Cílem tohoto přístupu je dát jakémukoliv klientovi třídy, tedy programátorovi používajícímu její rozhraní, záruku, že pokud získá referenci na tento objekt, bude vždy ve stavu připraveném pro použití a nemůže se mu stát, že při volání některé z metod obdrží chybu, že zapomněl něco nastavit. Myšlenkou je, že programátor používající tuto třídu musí pro její použití vidět pouze konstruktor a následně její rozhraní a nemusí pátrat, zda musí po vytvoření objektu něco nastavovat. Kromě toho tento přístup výrazně zjednodušuje testovatelnost třídy, jelikož v rámci Android testů běží všechny testy v jednom virtuálním stroji, mají tedy sdílenou paměť, což znamená, že statický stav zůstává sdílen mezi testy. Sdílený stav rozbíjí filozofii izolovanosti testů a může vyústit v situaci, kde jeden test selže, jelikož nějaký jiný test před ním nastavil něco do globálního stavu.

Často je však tuto filozofii poměrně složité následovat, jelikož aby mohl programátor vytvořit objekt, který v dané situaci potřebuje, musí mít přístup ke všem jeho závislostem, což může být obtížné zajistit a vyvstává diskuze, kde odkazy na tyto objekty držet, jak je zpřístupnit všem potenciálním klientům třídy a jak toto udělat přehledně bez toho, aniž by bylo vše vytvářeno na jednom místě. Toto může vyústit v techniky jako je lokátor služeb, již zmíněný Jedináček a podobné, kdy si třída sama ve své implementaci potřebnou závislost vytáhne, čemuž bychom chtěli předejít. Android nám tento problém také značně stěžuje tím, že hlavní komponenty aplikace jako aktivity, služby a objekt aplikace samotné jsou provázány co nejvolněji a nelze jednoduše předat jeden objekt z aktivity do aktivity, dokud se nejedná o primitivní typ, případně jej lze serializovat.

Tento problém pro nás řeší knihovny pro vkládání závislostí, které se nám snaží umožnit to, že si o potřebné závislosti prostě požádáme a knihovna nám je poskytne. My však musíme knihovně říct, jak tyto závislosti může získat a kde je má uložit. Dagger 2 používá objekt nazvaný `Component`, což je kontejner závislostí který dokáže vytvářet námi požadované objekty i s jejich závislostmi. Kontejner se skládá z modulů, které za podmínky určitých vstupů poskytnou kontejneru nové výstupy. V kódu se tyto třídy definují pomocí anotací a kód pro vytvoření kontejneru závislostí, nebo také objektového grafu, je vygenerován během kompilace a díky tomu se

za běhu vytvoří velmi rychle, což je jeden z hlavních důvodů, proč je Dagger 2 používá, protože za běhu vůbec nepoužívá reflexi, která je u knihoven tohoto typu častá a která způsobuje v Android aplikacích značné výkonnostní problémy. Knihovny pro vkládání závislostí existovaly již mnohem dříve, jako například Spring nebo Guice, při použití na platformě Android byly však díky reflexi pomalé a mohlo snadno dojít k chybě za běhu. Oproti tomu Dagger 2 vytváří objektový graf během kompilace, zkontroluje jestli je acyklický a všechny požadované závislosti je možné získat a v případě chyby nám neumožní aplikaci úspěšně zkompilovat, díky čemuž minimalizuje možné pády aplikace při nesprávném použití knihovny.

Dokumentaci a popis knihovny můžeme nalézt na http://google.github.io/dagger/ (10.4.2016).

2.3.5 Lombok

Lombok je další z nástrojů který rozšiřuje jazyk Java, respektive se nás snaží ušetřit obvyklých ceremonií jako implementace metod `equals, hashCode, toString`, implementaci přístupových metod k datům třídy a podobně. Moderní vývojová prostředí včetně Android Studia toto vše umí vygenerovat, nicméně při každé úpravě třídy se musí myslet na to, že je třeba příslušné metody vygenerovat znovu, aby zůstaly aktuální, což může vést k chybám.

Lombok pomocí anotací u příslušné třídy vygeneruje během kompilace obvyklých metod a díky Lombok pluginu do Android Studia nám vývojové prostředí tyto generované metody automaticky nabízí a díky tomu, že se metody generují během každé kompilace, generované metody zůstanou vždy aktuální. Jednoduchá datová třída se dvěma vlastnostmi z ukázky kódu 3 se všemi korektně napsanými implementacemi metod objektu `equals, hashCode, toString` a přístupových metod má délku kolem 50 řádků, nicméně díky tomuto nástroji lze totožnou funkcionalitu napsat jen na 5 řádků, jak vidíme na ukázce 3, navíc s benefitem, že pro přidání další vlastnosti do všech metod stačí přidat pouze řádek vlastnosti a promítne se korektně do všech dříve zmíněných metod.

```
@lombok.Data
public class Position {
  private double latitude;
  private double longitude;
}
```

Výpis 3: Datová třída používající Lombok

Nástroj navíc poskytuje další užitečné anotace, například `@Value` pro generování neměnitelných objektů bez přístupových metod, `@Builder` pro generování stavitelů objektů pro eliminaci konstruktorů s velkým počtem parametrů, případně `@SneakyThrows`, který nás zbavuje nutnosti obalovat kontrolované výjimky `try-catch` blokem případně výjimku propagovat do signatury metody a mnohé další.

Stránky Lomboku můžeme nalézt na https://projectlombok.org/ (10.4.2016).

2.3.6 Google Play Services

Soubor knihoven od Googlu, které se snaží vývojáři rozšířit možnosti implementace všemi různými způsoby. Obsahuje například knihovny pro rozpoznávání obličeje a nálady, plánování úloh, podporu monetizace aplikace, mnohé další a také námi používané rozhraní na upozornění přiblížení se k určitému místu a také Google Drive API.

Jedná se v podstatě o rozšiřující API celého frameworku, které by mělo být dostupné na každém zařízení, které obsahuje aplikaci obchodu Google Play a tudíž na všech zařízeních, kde je možné naší aplikaci stáhnout. Knihovna je zároveň přes aplikaci obchodu aktualizovaná a tudíž u ní nemusíme řešit problémy kompatibility, jako u vlastního SDK [7].

Přehled všech rozhraní lze nalézt na https://developers.google.com/android/guides/overview (10.4.2016).

2.3.7 Butter Knife

Jedná se o knihovnu, která nás zbavuje dalších ceremonií Androidu nastavování posluchačů na události objektů uživatelského rozhraní, jako klepnutí, změnu textu a podobně. Knihovna pomocí anotací nad požadovanými metodami tyto posluchače a jejich nastavování generuje a my těmito anotacemi pouze specifikujeme, co se má během dané události stát. Ukázku zobrazování nápovědu u obrázků, která je používána v aplikaci, můžeme vidět na ukázce 4.

```
@OnLongClick(R.id.navigate) boolean showContentDescription(View v){
  return ViewUtils.showContentDescription(v);
}
```

Výpis 4: Použití knihovny Butter Knife

Knihovnu můžeme najít na http://jakewharton.github.io/butterknife/ (10.4.2016).

2.3.8 DBFlow

Jedné se o knihovnu pro objektově relační mapování objektů pro databáze SQLite, které se v Androidu používají jako výchozí. Práce s touto databází pomocí API Androidu není složitá, nicméně kód je poměrně zdlouhavý a pro přečtení nebo uložení objektu je potřeba mnoho řádek kódu. Pro objektově relační mapování existuje mnoho knihoven, najít mezi nimi však takovou, která by vyhověla všem požadavkům, je velice obtížné. Velká většina těchto knihoven využívá reflexi, která svou časovou náročností může překonat i operace zápisu a čtení z databáze. DBFlow se snaží tento problém řešit generací kódu pro čtení a zápis dat pomocí anotací nad objekty a jejich vlastnostmi, následně jejich ukládání řeší pomocí adaptérů vygenerovaných pro každou anotovanou třídu a SQL dotazy se píší pomocí generovaných metod a metod knihovny. DBFlow není také ideální a nutí vývojáře používat jejich bázovou třídu pro mapované objekty, případně implementovat vlastní adaptér a některé databázové vazby se pomocí anotací nedefinují nejlépe. Pro naše řešení však tato knihovna zcela vyhovuje a vygenerovaný kód práce s databází se

velice podobá tomu, který by vývojář sám musel psát. Ukázka sestavování SQL dotazů pomocí DBFlow můžeme vidět na ukázce 5.

```
RouteData select(long id){
  return SQLite.select().from(RouteData.class)
          .where(RouteData_Table._id.eq(id)).querySingle();
}
```

Výpis 5: Sestavování SQL dotazů s DBFlow

DBFlow nám také umožňuje přidávat adaptéry pro vlastní datové typu a tím nám umožnit uložit typ do jednoho sloupce, kde by si bez adaptéru knihovna neuměla poradit. V aplikaci tohoto využíváme pro uložení identifikátorů `UUID` a také objektů `LatLng`, které obsahují GPS souřadnice a adaptér je ukládá jako text oddělený čárkou. Knihovna obsahuje také některé vestavěné adaptéry pro často potřebné objekty, například námi také využívaný objekt `Date`.

Další podporovanou funkcí jsou migrace, kde se opět pouze anotacemi dá definovat třída, která má migraci databáze na novou verzi převést.

Knihovnu můžeme najít na https://github.com/Raizlabs/DBFlow (10.4.2016).

2.3.9 Robolectric

Jednotkové testy využívající kód Androidu je poměrně zdlouhavé spouštět, neboť kód běží na zařízení a pro každé spuštění testu se musí sestavit nové testovací APK, nahrát na zařízení a až poté spustit testy. Pokud se programátor pokusí spustit testy na svém počítači s využitím SDK Androidu, obdrží při prvním použití výjimku, neboť SDK obsahuje pouze deklarace metod bez implementací a skutečné implementace jsou pouze na zařízení.

Robolectric se snaží toto řešit pomocí emulace SDK Androidu na počítači a umožnit tak vývojáři testovat svůj kód bez nahrávání na zařízení a výrazně tak zrychlit testování. Robolectric používá tzv. stínové objekty pro všechny třídy SDK a pomocí nich lze emulovat systémové události, získávat systémové objekty a případně nahrazovat některé vlastní implementací pro simulaci určitých podmínek. Knihovna se vyvíjí už mnoho let a díky záměru simulovat celý framework je velmi rozsáhlá a zatím stále ne zcela dokonalá, nicméně se stala standardem pro testování Android kódu přímo na pracovní stanici.

Knihovnu lze nalézt na http://robolectric.org/ (10.4.2016).

2.3.10 Další použité knihovny

- **Universal Image Loader** - Práce s obrázky a fotkami. (https://github.com/nostra13/Android-Universal-Image-Loader (10.4.2016))
- **Timber** - Logování zpráv v aplikaci. (https://github.com/JakeWharton/timber (10.4.2016))
- **Okio** - Efektivní a snadné práce se soubory a datovými proudy. (https://github.com/square/okio (10.4.2016))

- **EventBus** - Snadné publikování a konzumování globálních události v aplikaci. (http://greenrobot.org/eventbus/ (10.4.2016))
- **ShowcaseView** - Zobrazení uživatelsky přívětivé nápovědy. (https://github.com/amlcurran/ShowcaseView (10.4.2016))
- **JUnit 4** - Známý framewrok pro Java jednotkové testy. (http://junit.org/ (10.4.2016))
- **AssertJ Android** - Pomocné aserce pro jednotkové testy. (http://square.github.io/assertj-android/ (10.4.2016))
- **Mockito** - Mockovací knihovna pro jednotkové testy. (http://mockito.org/ (10.4.2016))

3 Návrh aplikace

V této kapitole si napřed uvedeme, jak probíhalo vytvoření zadání pro aplikaci, poté představím scénáře využití aplikace a návrh uživatelského rozhraní. V závěru kapitoly jsou popsány další nástroje, které byly pro návrh využity.

Mentální postižení není nemoc, je to trvalý stav, který se projeví do 18 let věku a nedá se léčit [10]. U každého člověka s mentálním postižením se jeho stav projevuje jinak a je proto potřeba k nim přistupovat individuálně. Tito lidé mají často potíže s učením, porozuměním, sníženou míru pozornosti a horší orientaci v čase a prostoru [10], což je pro spolehlivé cestování důležité. Aplikace byla navrhována s přihlédnutím k těmto faktům a v každé její části jsme se snažili o co nejjednodušší vzhled aplikace a její ovládání.

3.1 Příprava zadání

Na začátku práce bylo potřeba stanovit si cíle a zjistit, jaká úskalí vlastně lidé s mentálním postižením při cestování zažívají, s jakými problémy se potýkají a co by pro ně mohlo být přínosné.

Úvodní konzultace byly vedeny s paní Mgr. Barborou Uhlířovou z Brněnského poradenského centra Společnosti pro podporu lidí s mentálním postižením v České republice, z.s. Od paní Uhlířové byly získány cenné rady, jak postupovat a zdroje, které mi nám pomohly lépe pochopit, jak se k mentálně postiženým chovat, co od nich očekávat a jak k nim, jako k uživatelům mobilní aplikace přistupovat.

Po prvních konzultacích byly vytvořeny návrhy, co by mohla aplikace dělat a tyto návrhy byly diskutovány, kdy paní Uhlířová přibližovala situaci klientů, jejich problémy s orientací a se soustředěním při cestování. Po další emailové komunikaci bylo s paní Uhlířovou dohodnuto, že pro lepší zadání bude vše dále konzultováno s více kolegy z praxe v Brně.

Setkání se zúčastnilo množství asistentů mentálně postižených, kterým byly existující návrhy prezentovány. Asistenti měli k těmto návrhům mnoho připomínek z praxe, navrhli mnoho rozšíření založené na zkušenostech s klienty, jako nutnost připomínání povinností, upozorňování a podobně. Následně byli definováni aktéři aplikace a vytvořena sada případů užití, které dávají pro tuto aplikaci smysl a mnohé z nich budou implementovány.

3.2 Popis aktérů

Obrázek 2: Aktéři systému

Pro uživatele klienta i asistenta jsou definovány různé případy užití, neboť se celé chování a použití aplikace bude v obou případech značně lišit. Grafické zobrazení aktérů je na obrázku 2. Pro ilustraci případů užití jednotlivých aktérů jsou použity obrázky z reálné aplikace pro lepší přiblížení a popis daného případu užití.

3.2.1 Uživatel Asistent

Jedná se o aktéra, který je zároveň klientovi s mentálním postižením odborným asistentem, jenž o něj pečuje a poskytuje mu podporu. Tento aktér je relevantní k vytváření obsahu, který má pomoci klientovi k lepší orientaci při cestování a také mu může připravenou cestu předvést. Může také nahraná data editovat a případně je rozšiřovat. Může také zadat do aplikace své kontaktní údaje pro možnou nečekanou situaci klienta na cestách, případně nastavit zálohování uložených dat pro zamezení jejich ztráty při ztrátě telefonu nebo jeho výměně za jiný.

3.2.2 Uživatel Klient

Klient je osoba pro kterou je aplikace primárně určena a má mu pomoci vyřešit problém, v tomto případě pomoci z orientací při cestování. Může si prohlížet obsah a zejména ho využívat při cestách v terénu. Uživatel by měl být upozorněn na všechny uložené a rozeznané data v závislosti na své pozici a měl by tak získat relevantní informace k tomu, kde se právě nachází. Dále může aplikaci využít ke snadnému kontaktování svého asistenta.

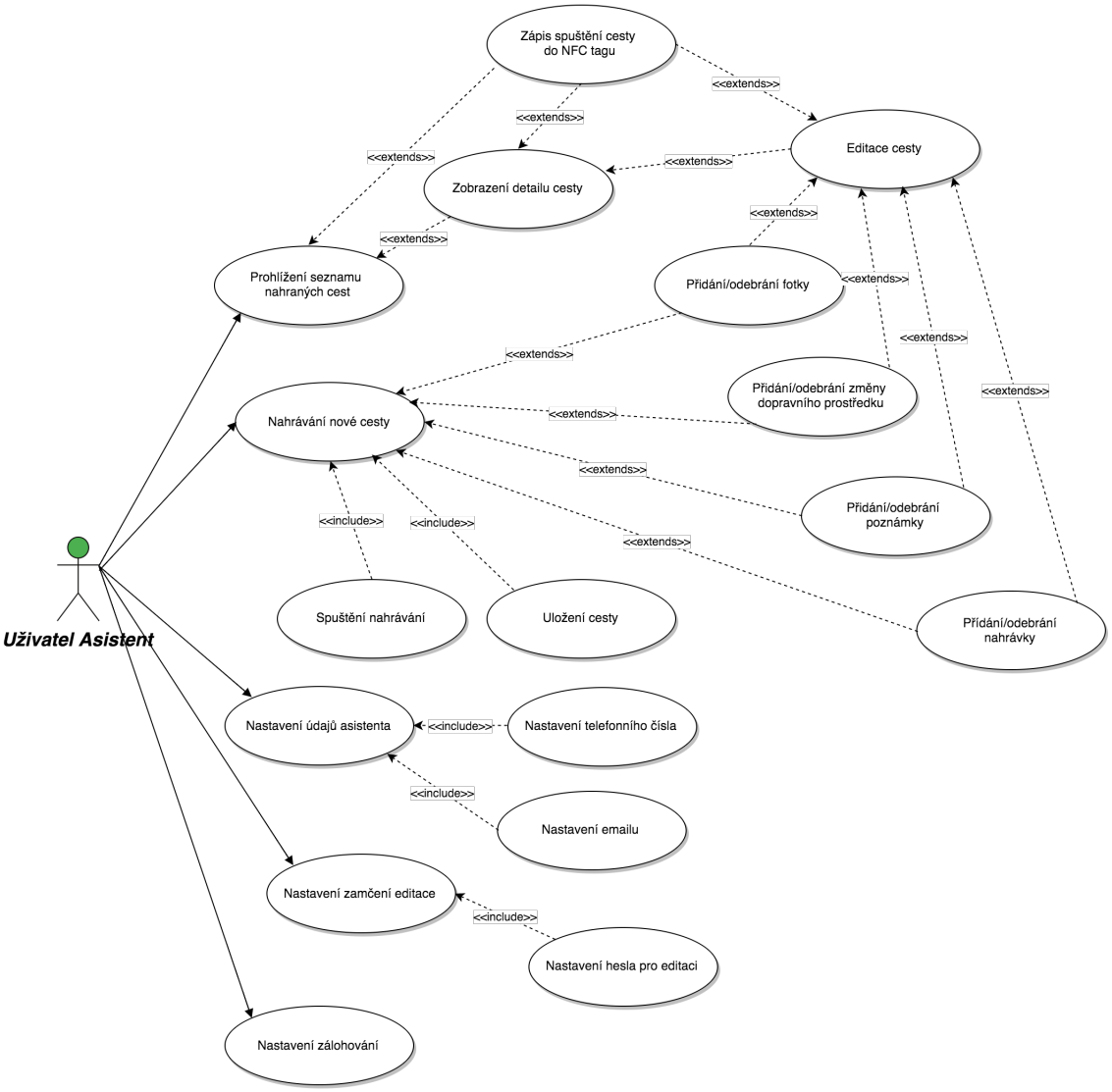
3.2.3 Uživatel Administrátor

Administrátor může být spravující asistent, případně jiná osoba, která je oprávněna spravovat data nahraná ostatními asistenty a využívána klienty. Může tyto záznamy prohlížet a případně upravovat a ovlivnit tak, co se kterému uživateli zobrazí a jakým způsobem bude asistence probíhat.

3.3 Případy užití uživatelů

Pro uživatele asistenta jsou určeny složitější operace pro vytváření interaktivního obsahu pro klienta, nastavování aplikace a prezentace klientovi. Pro případy užití asistenta platí, že klient může těmto krokům přihlížet, pokud o to projeví zájem. Případy užití asistenta a jejich vztahy lze vidět na obrázku 3.

Pro klienta jsou určeny více intuitivní a nenáročné operace vyžadující co nejméně aktivních kroků z klientovi strany. Aplikace by měla na základě polohy a dalších údajů sama rozpoznat, co má v danou chvíli udělat. Případy užití klienta a jejich vztahy lze vidět na obrázku 8.



Obrázek 3: Diagram případů užití uživatele asistent

3.3.1 Prohlížení seznamu nahraných cest

Aktéři: Asistent

Hlavní scénář: Na úvodní obrazovce jsou zobrazeny všechny dosud nahrané cesty v seznamu pod sebou. Lze vidět pouze základní údaje a přiřazený obrázek pro snadnou orientaci. Uživatel se pomocí kliknutí na řádek může podívat na celý detail cesty, případně pomocí rychlých akcí spustit ihned asistenci a podobně. Obrazovku prohlížení cest lze vidět na obrázku 6.

Spouštěč: Uživatel spustí aplikaci nebo se do ní vrátí pomocí notifikace v liště upozornění.

Rozšíření:

- Nahrávání cesty

- Zobrazení detailu cesty
- Zapsání cesty do NFC tagu

3.3.2 Nahrávání cesty

Aktéři: Asistent

Hlavní scénář: Uživateli se zobrazí nahrávací okno, kde může vidět svou aktuální pozici a po klepnutí na tlačítko nahrávat začne aplikace sbírat data o poloze a umožní mu přidávat další data k právě nahrávané pozici. V notifikační liště se objeví upozornění o tom, že aplikace právě nahrává. Uživatel může v tu chvíli odejít z aplikace, případně navštívit její jiné obrazovky a poté se do nahrávání vrátit, aniž by ztratil právě nahrávaná data. Stejně tak po opuštění aplikace při nahrávání se do ní může vrátit klepnutím na zobrazenou notifikaci.

Na mapě vidí uživatel dosud nahranou a uloženou cestu a případně přidaná data. Klepnutím na tlačítko uložit se aktuálně nahraní data a média uloží, případně aktualizují. Pokud uživatel dokončil nahrávání cesty, klepne na tlačítko uložit a poté tlačítkem stop zastaví nahrávání, notifikace zmizí a cesta je v seznamu cest připravena pro spuštění asistence, případně následnou editaci. Pokud uživatel ukončí nahrávání bez uložení, je na toto upozorněn a při potvrzení ukončení jsou nahraná data smazána. Obrazovku nahrávání můžeme vidět na obrázku 4 a zobrazenou notifikaci na obrázku 5.

Prekondice: Uživatel má v telefonu povolené získávání polohy pomocí GPS.

Spouštěč: Uživatel klepne na tlačítko nahrávat v seznamu nahraných cest.

Rozšíření:

- Přidání fotky
- Přidání zvukové nahrávky
- Přidání poznámky
- Přidání změny dopravního prostředku
- Odebrání fotky
- Odebrání zvukové nahrávky
- Odebrání poznámky
- Odebrání změny dopravního prostředku

3.3.3 Zobrazení detailu cesty

Aktéři: Asistent

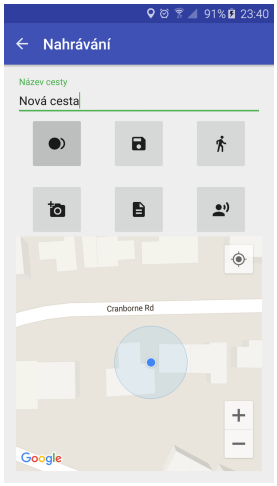
Hlavní scénář: Uživateli se zobrazí detail cesty se všemi informacemi, které jsou o ní uloženy. Na mapě si může prohlédnout kudy vedla, na místech pořízení fotografií jsou miniatury fotek, při změně dopravních prostředků lze vidět ikony daných prostředků, zvukově a textové záznamy jsou označeny příslušnou ikonou. Při klepnutí na indikátor některého z těchto záznamů se zobrazí jeho popis, který uživatel dříve zadal. V detailu cesty lze poklepáním na tlačítko editovat přejít do módu editace cesty a všechny dříve zmíněné záznamy upravit. Ukázku lze vidět na obrázku 7.

Prekondice: Cesta je uložena.

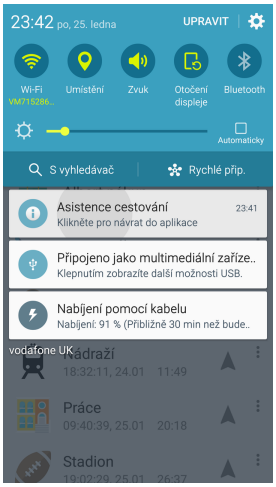
Spouštěč: Uživatel klepne na řádek se zobrazenou cestou v seznamu cest.

Rozšíření:

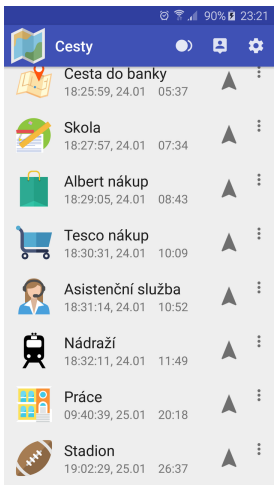
- Editace cesty
- Zapsání cesty do NFC tagu



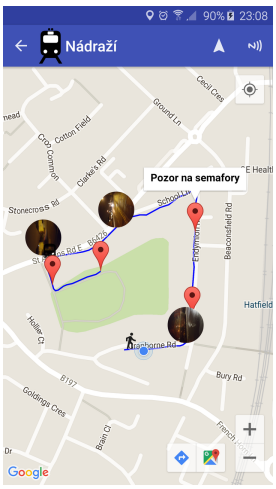
Obrázek 4: Nahrávání cesty



Obrázek 5: Zobrazená notifikace



Obrázek 6: Prohlížení seznamu cest



Obrázek 7: Zobrazení detailu cesty

3.3.4 Editace cesty

Aktéři: Asistent

Hlavní scénář: Uživatel může editovat všechny uložené informace o cestě. Přepsání editačních polí může změnit názvy a popis nahrané cesty, podržením a přetažením miniatur fotek nebo ikon dalších údajů na mapě může změnit jejich pozici a tudíž místo, kdy se později vyvolají. Dále může podržením a přetažením měnit tvar trasy, při klepnutí na mapu přidat další záznamy. Jakmile je uživatel s editací spokojen, klepnutím na tlačítko uložit se údaje zapíší do databáze a následující asistence cestování bude tyto údaje používat.

Prekondice: Cesta je uložena.

Spouštěč: Uživatel klepne na tlačítko editovat v detailu cesty.

Rozšíření:

- Přidání fotky
- Přidání zvukové nahrávky
- Přidání poznámky
- Přidání změny dopravního prostředku
- Odebrání fotky
- Odebrání zvukové nahrávky
- Odebrání poznámky
- Odebrání změny dopravního prostředku

3.3.5 Prohlížení seznamu nahraných cest klientem

Aktéři: Klient

Hlavní scénář: Na úvodní obrazovce jsou pod sebou vyobrazeny všechny cesty, pro které může uživateli aplikace poskytnout asistenci. Uživatel si prohlíží cesty, které mají zobrazeny pouze základní informace a jsou označeny vybraným obrázkem pro snazší orientaci. Uživatel si poklepnutím na řádek cesty může dostat na její detail případně rychlými akcemi spustí navigaci.

Spouštěč: Uživatel klepne na ikonu aplikace v telefonu.

Rozšíření:

- Detail nahrané cesty pro klienta
- Asistence cestování

3.3.6 Detail nahrané cesty pro klienta

Aktéři: Klient

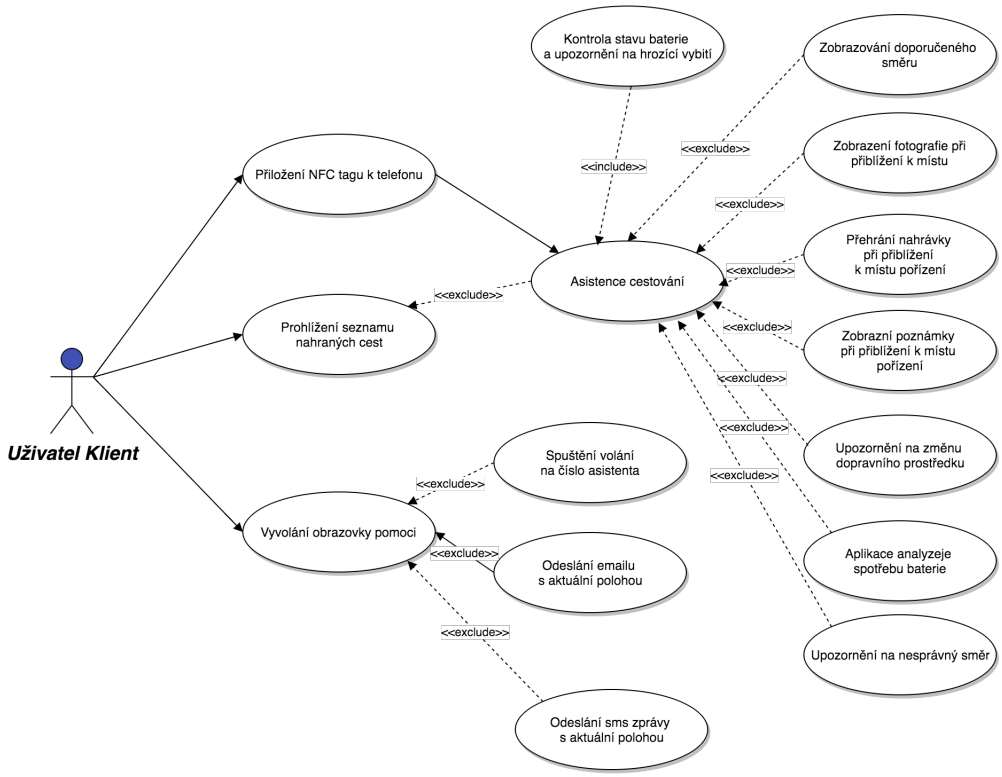
Hlavní scénář: Klientovi se zobrazí obrazovka se všemi údaji, které mu mají na cestě pomoci. Fotky jsou vidět jako miniatury na místě, kde byly pořízeny, podobně nahrávky, změny dopravního prostředku a poznámky, které jsou reprezentovány svými příslušnými jednoduchými ikonami.

Prekondice: Uživatel má nahranou cestu.

Spouštěč: Uživatel klepne na řádek cesty v seznamu nahraných cest.

Rozšíření:

- Asistence cestování



Obrázek 8: Diagram případů užití uživatele klient

3.3.7 Zapsání cesty do NFC tagu

Aktéři: Asistent

Hlavní scénář: Uživateli se zobrazí obrazovka, která jej instruuje k přiložení NFC tagu k telefonu. Jakmile telefon zaznamená blízkost tagu, zapíše do něj informaci o uložené cestě pro následné rychlé spouštění při přiložení tagu k telefonu. Obrazovku zápisu do NFC tagu lze vidět

na obrázku 9 a oznámení o zapsaném tagu na obrázku 10. Technologie NFC je blíže popsána v sekci 4.8.

Prekondice: Cesta je uložena.

Spouštěč: Uživatel klepne na ikonu NFC v detailu cesty nebo na rozšiřující menu v seznamu cest.

3.3.8 Spuštění asistence cestování po přiložení NFC tagu k telefonu

Aktéři: Klient

Hlavní scénář: Klient přiloží telefon k nálepce, kartě nebo čemukoliv jinému, co bylo použito jako NFC tag a obsahuje informace o cestě. Tato informace se přečte a spustí se aplikace se zapnutou asistencí cestování na cestu zapsanou na tagu.

Prekondice: Klient má k dispozici NFC tag, na který byla dříve zapsána informace o cestě.

Spouštěč: Klient přiloží telefon k NFC tagu.

Obrázek 9: Zápis do NFC tagu

Obrázek 10: Zapsáno do NFC tagu

3.3.9 Asistence cestování

Aktéři: Klient

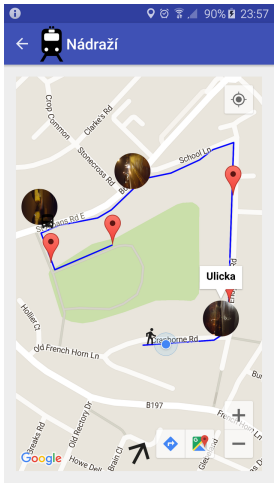
Hlavní scénář: Uživateli se zobrazí jednoduchá mapka s vyznačenou cestou, kterou má absolvovat. Zobrazí se mu všechny dostupné informace jako čas do cíle, doporučený směr, nejbližší nahrané fotky, nahrávky nebo poznámky. Zkontroluje se podle uložených statistik, zda bude uživateli stačit baterie k absolvování cesty a pokud ne, bude na to upozorněn. Aplikace využívá klientovu polohu pro zobrazení relevantních nahraných informací a snaží se mu napovídat následující směr pomocí zobrazené šipky. Jak se klient pohybuje, aplikace mu poskytuje nahraný obsah asistentem, který by mu měl pomoci v danou chvíli se lépe zorientovat. Obrazovku asistence cestování lze vidět na obrázku 11.

Prekondice: Klient má nahranou cestu a spustil asistenci cestování po této cestě.

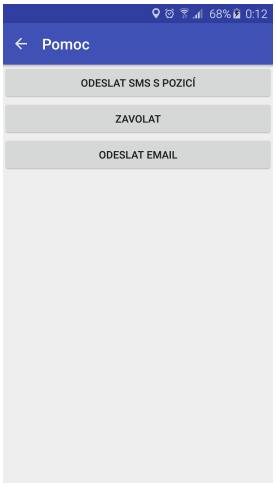
Spouštěč: Klient klepl na obrázek navigovat.

Rozšíření:

- Zobrazení fotky
- Přehrání uložené zvukové nahrávky
- Zobrazení uložené poznámky
- Upozornění na změnu dopravního prostředku
- Analýza spotřeby baterie
- Zobrazení doporučeného směru
- Upozornění na nesprávný směr



Obrázek 11: Asistence cestování



Obrázek 12: Obrazovka pomoci

3.3.10 Zobrazení doporučeného směru

Aktéři: Klient

Hlavní scénář: Na základě pohybu uživatele jsou data vyhodnocována a vypočítán aktuální směr uživatele. Ten se porovná s očekávaným a uloženým směrem a podle rozdílu se zobrazí směr, kterým by se uživatel měl ideálně vydat. Informace je zobrazena v dolní části obrazovky v podobě jednoduché šipky.

Prekondice: Uživatel se pohybuje po cestě nebo chodníku.

Spouštěč: Uživatel využívá asistence při cestování.

3.3.11 Upozornění na nesprávný směr

Aktéři: Klient

Hlavní scénář: Když se uživatel nepohybuje dle očekávané trasy, aplikace může vyhodnotit tento pohyb jako nesprávný. V takovém případě začne uživatele upozorňovat a pokud směr nezmění nebo neklepne na tlačítko známý směr, bude s upozorňováním pokračovat a nabízet směr správný.

Spouštěč: Uživatel se během asistence cestování určitou dobu pohybuje jiným, než očekávaným směrem.

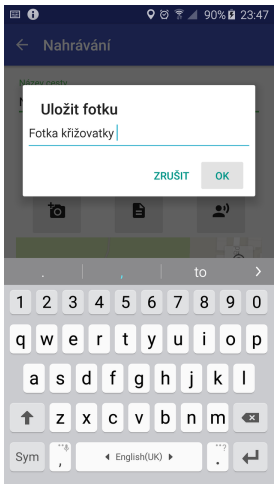
3.3.12 Přidání fotky

Aktéři: Asistent

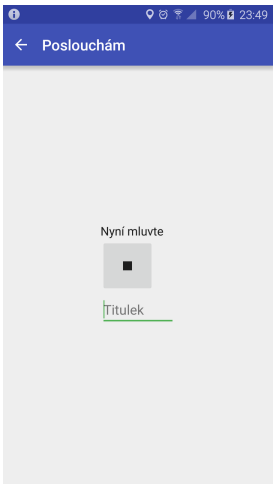
Hlavní scénář: Spustí se fotoaparát zařízení a uživatel může začít fotit. Jakmile pořídí fotografii, zobrazí se v aplikace okno pro zadání názvu fotky s dotazem pro uložení. Uživatel zadá název a klepnutím na potvrzovací tlačítko se fotka uloží mezi data aplikace a zobrazí mezi aktuálními fotografiemi na místě polohy uživatele v době pořízení fotografie. Uložení fotky lze vidět na obrázku 13.

Prekondice: Uživatel nahrává nebo edituje cestu.

Spouštěč: Uživatel klepl na tlačítko přidat fotku.



Obrázek 13: Pojmenování fotky po pořízení



Obrázek 14: Přidání zvukové nahrávky

3.3.13 Odebrání fotky

Aktéři: Asistent

Hlavní scénář: Uživateli se zobrazí okno s dotazem, zda chce fotku odebrat. Po klepnutí na potvrzovací tlačítko je fotka smazána z úložiště a odstraněna její miniatura z aktuální cesty.

Prekondice: Uživatel nahrává nebo edituje cestu a má uloženou fotografii.

Spouštěč: Uživatel podrží prst na miniatuře fotografie a vybere možnost smazat.

3.3.14 Zobrazení fotky

Aktéři: Klient

Hlavní scénář: Uživatel se přiblíží k místu pořízení fotografie, aplikace toto rozpozná a upozorní uživatele na přítomnost fotky, která se zobrazí na dobu, po kterou je uživatel nablízku. Přes spodní část fotky je zobrazen popisek, který této fotce dříve zadal asistent.

Prekondice: Uživatel má k dané cestě uloženou fotku.

Spouštěč: Uživatel se přiblíží místu pořízení fotky nebo klepne na miniaturu fotky.

3.3.15 Přidání zvukové nahrávky

Aktéři: Asistent

Hlavní scénář: Uživateli se zobrazí obrazovka s textem mluvte a zvuk se zaznamenává. Když je uživatel hotov, klepne na tlačítko stop a může nahrávce přidat název, který se bude později zobrazovat. Poté tlačítkem nahrávku uloží mezi data aplikace s přiřazenou aktuální polohou uživatele. Zobrazení nahrávání lze vidět na obrázku 14.

Prekondice: Uživatel nahrává nebo edituje cestu.

Spouštěč: Uživatel klepl na tlačítko přidat nahrávku.

3.3.16 Odebrání zvukové nahrávky

Aktéři: Asistent

Hlavní scénář: Uživateli se zobrazí dotaz, zda chce nahrávku opravdu smazat, po potvrzení se uložená nahrávka odstraní a zmizí její ikona.

Prekondice: Uživatel nahrává nebo edituje cestu a má uloženou nahrávku.

Spouštěč: Uživatel podrží prst na ikoně nahrávky a vybere možnost smazat.

3.3.17 Přehrání uložené zvukové nahrávky

Aktéři: Klient

Hlavní scénář: Uživatel se zapnutou asistencí cestování přiblíží místu, kde byla dříve za pomoci asistenta pořízena zvuková nahrávka. Uložená nahrávka se načte a začne se uživateli přehrávat. Zároveň se zobrazí popis nahrávky a tlačítko umožňující nahrávku přehrát znovu.

Prekondice: Uživatel má k dané cestě uloženou zvukovou nahrávku.

Spouštěč: Uživatel se přiblíží k místu pořízení nahrávky nebo klepne na ikonu nahrávky.

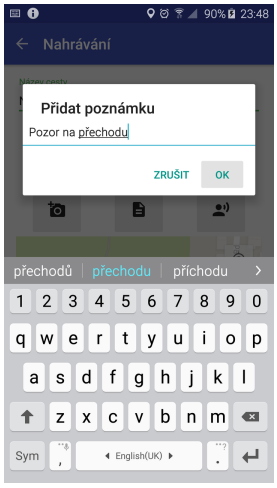
3.3.18 Přidání poznámky

Aktéři: Asistent

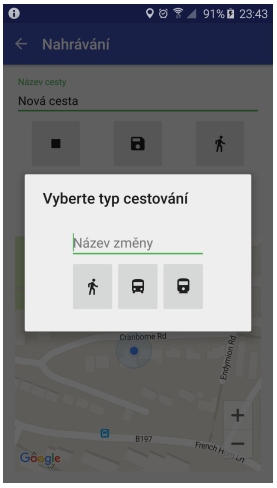
Hlavní scénář: Uživateli se zobrazí okno s textovým vstupem a vysune se klávesnice. Uživatel zadá textovou poznámku a po klepnutí na potvrzovací tlačítko se poznámka uloží s asociací k aktuální poloze uživatele. Okno přidávání poznámky lze vidět na obrázku 15.

Prekondice: Uživatel nahrává nebo edituje cestu.

Spouštěč: Uživatel klepl na tlačítko přidat poznámku.



Obrázek 15: Přidání poznámky



Obrázek 16: Změna dopravního prostředku

3.3.19 Odebrání poznámky

Aktéři: Asistent

Hlavní scénář: Uživateli se zobrazí okno s dotazem na smazání poznámky. Po potvrzení se nahrávka smaže a zmizí ikona indikující nahrávku.

Prekondice: Uživatel nahrává nebo edituje cestu a má uloženou poznámku.

Spouštěč: Uživatel podrží prst na ikoně poznámky a vybere možnost smazat.

3.3.20 Zobrazení uložené poznámky

Aktéři: Klient

Hlavní scénář: Uživatel je při přiblížení se místu poznámky upozorněn a poznámka se zobrazí.

Prekondice: Uživatel má k dané cestě uloženou poznámku.

Spouštěč: Uživatel se přiblíží k místu pořízení poznámky nebo klepne na ikonu poznámky.

3.3.21 Přidání změny dopravního prostředku

Aktéři: Asistent

Hlavní scénář: Zobrazí se okno s nabídkou tlačítek s obrázkem dopravního prostředku. Uživatel může zároveň přidat ke změně dopravního prostředku popisný text. Po kliknutí na tlačítko s dopravním prostředkem se změna dopravního prostředku uloží s asociovanou aktuální polohou uživatele. Okno změny dopravního prostředku lze vidět na obrázku 16.

Prekondice: Uživatel nahrává nebo edituje cestu.

Spouštěč: Uživatel klepl na tlačítko s aktuálním dopravním prostředkem.

3.3.22 Odebrání změny dopravního prostředku

Aktéři: Asistent

Hlavní scénář: Uživateli se zobrazí okno s dotazem na smazání změny dopravního prostředku. Po potvrzení se nahrávka smaže a zmizí ikona indikující nahrávku. Zobrazené nastavení údajů lze vidět na obrázku 17.

Prekondice: Uživatel nahrává nebo edituje cestu a má uloženou změnu dopravního prostředku.

Spouštěč: Uživatel podrží prst na ikoně změny dopravního prostředku a vybere možnost smazat.

3.3.23 Upozornění na změnu dopravního prostředku

Aktéři: Klient

Hlavní scénář: Uživateli je při přiblížení ke změně dopravního prostředku upozorněn, zobrazí se mu obrázek dopravního prostředku, do kterého má nastoupit nebo naopak vystoupit spolu s textem, který byl při nahrávání cesty zadán.

Prekondice: Uživatel má uloženo upozornění na změnu dopravního prostředku.

Spouštěč: Uživatel se přiblíží k místu změny dopravního prostředku.

3.3.24 Nastavení údajů asistenta

Aktéři: Asistent

Hlavní scénář: Uživateli se zobrazí obrazovka s nastavením údajů, kde může vyplnit své údaje.

Spouštěč: Uživatel klepl na tlačítko nastavení.

Rozšíření:

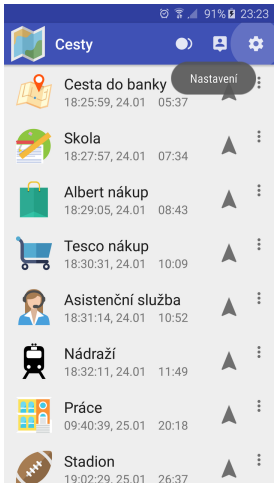
- Nastavení telefonního čísla asistenta
- Nastavení emailu asistenta

3.3.25 Nastavení telefonního čísla asistenta

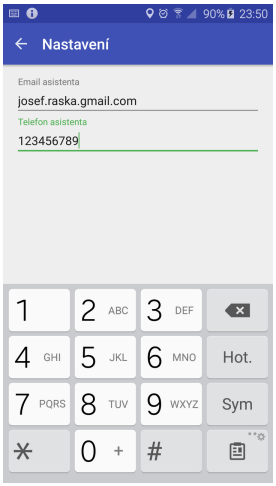
Aktéři: Asistent

Hlavní scénář: Editovatelné pole pro telefonní číslo se zvýrazní a vysune se numerická klávesnice. Uživatel zadá své telefonní číslo, které se automaticky formátuje do přívětivějšího formátu. Po ukončení editace je telefonní číslo automaticky uloženo. Pokud telefonní číslo neodpovídá formátu telefonních čísel v České republice, uživatel je na toto upozorněn a je vyzván k úpravě vstupu. Zadávání telefonního čísla vidíme na obrázku 18.

Spouštěč: Uživatel klepl na editační pole s telefonním číslem asistenta.



Obrázek 17: Spuštění nastavení údajů



Obrázek 18: Nastavení telefonního čísla

3.3.26 Nastavení emailu asistenta

Aktéři: Asistent

Hlavní scénář: Editovatelné pole pro emailovou adresu se zvýrazní a vysune se klávesnice. Uživatel zadá svou emailovou adresu. Po ukončení editace je emailová adresa automaticky uložena. Pokud emailová adresa neodpovídá správnému formátu emailové adresy, uživatel je na toto upozorněn a vyzván k úpravě vstupu.

3.3.27 Vyvolání obrazovky pomoci

Aktéři: Klient

Hlavní scénář: Uživateli se zobrazí obrazovka s volbami rychlých akcí, které kontaktují jeho asistenta. Obrazovka pomoci je na obrázku 12.

Prekondice: V zařízení jsou uloženy kontaktní údaje asistenta.

Spouštěč: Uživatel klepne na obrázek pomoc v aplikaci.

Rozšíření:

- Volání asistentovi
- Poslání SMS zprávy s aktuální polohou

- Poslání emailu s aktuální polohou

3.3.28 Volání asistentovi

Aktéři: Klient, Asistent

Hlavní scénář: Uživateli se zobrazí obrazovka, kde vidí vytáčení čísla na svého asistenta. Hovor se poté uskuteční.

Prekondice: V aplikaci je uloženo telefonní číslo asistenta.

Spouštěč: Uživatel klepne na číslo asistenta.

3.3.29 Poslání SMS zprávy s aktuální polohou

Aktéři: Klient, Asistent

Hlavní scénář: Uživateli se zobrazí potvrzení o odeslání SMS zprávy a její znění. Asistentovi se zobrazí zpráva, ve které je také odkaz na mapu s GPS polohou klienta v době odesílání zprávy.

Prekondice: V aplikaci je uloženo telefonní číslo asistenta.

Spouštěč: Uživatel klepne na tlačítko poslat SMS.

3.3.30 Poslání emailu s aktuální polohou

Aktéři: Klient, Asistent

Hlavní scénář: Klientovi se otevře emailová aplikace s nachystanou zprávou a adresou asistenta a klient klepne odeslat. Asistentovi přijde email s žádostí o pomoc a odkazem na mapu s GPS polohou klienta v době odeslání emailu.

Prekondice: V aplikaci je uložena emailová adresa asistenta.

Spouštěč: Uživatel klepne na tlačítko odeslat email.

Spouštěč: Uživatel klepl na editovatelné pole emailové adresy.

3.3.31 Nastavení zamčení editace

Aktéři: Asistent

Hlavní scénář: Uživateli se zobrazí editovatelné pole, kde může zadat heslo pro zamčení editace obsahu aplikace. Pokud toto heslo nezadá, aplikace bude stále v módu editace a uložená data bude možné upravit. Uživatel zadá heslo, to bude uloženo a bude nyní vyžadováno pro zpřístupnění úprav cest, případně nastavení.

Spouštěč: Uživatel klepl na tlačítko nastavení zamčení.

3.3.32 Nastavení zálohování

Aktéři: Asistent

Hlavní scénář: Uživateli se zobrazí výběr z listu, zda chce zálohovat manuálně, jednou za den, týden nebo za měsíc. Uživatel vybere jednu z možností a podle výběru se naplánují příslušné akce.

Prekondice: Uživatel má na zařízení nastaven účet umožňující zálohování.

Spouštěč: Uživatel klepl na tlačítko nastavení zálohování.

3.3.33 Analýza spotřeby baterie

Aktéři: Klient

Hlavní scénář: Během cestování se ukládají průběžně data o spotřebě baterie a délce trvání cesty. Data jsou poté spolu s předchozími záznamy statisticky analyzována a je odhadována a upřesňována očekávaná výdrž baterie během navigace. Na základě těchto analýz může být uživatel poté upozorněn na nebezpečí vybití baterie v terénu.

Spouštěč: Uživatel využívá asistence při cestování.

3.4 Grafický návrh

Cílem návrhu bylo používat v aplikaci konzistentní rozhraní a také jednoduchost všech uživatelských prvků pro snadné pochopení pro uživatele se specifickými nároky. Většina uživatelských prvků je tak realizována jako velká tlačítka, která při podržení prstu na prvku zobrazí jeho textový popis. Jako hlavní směr návrhu uživatelského rozhraní byl zvolen Material design [6], neboť se jedná o aktuální doporučení pro mobilní aplikace a je v něm kladen velký důraz na jednoduchost a konzistenci, tedy naše požadavky. Velkou část této specifikace za nás řeší námi používaná knihovna Android Support, popsaná v sekci 2.3.1.

3.4.1 Material design

Úplnou specifikaci Material designu, vydanou poprvé v roce 2015 lze nalézt v [6]. Jedná se o detailní popis všech grafických prvků uživatelského rozhraní, které by měl vývojář používat, jejich chování, vzhled i vzájemnou interakci. Jako hlavní cíl si klade vytvořit vizuální jazyk, spojující klasické principy dobrého designu a nových možností moderních technologií a vyvinout systém spojující zkušenost uživatele na různých platformách a velikostech obrazovek [6]. Uživatelské rozhraní v duchu Material designu by mělo uživateli lépe napovídat, co který prvek dělá, pomocí animací a směru efektů přibližovat význam grafických komponent, případně pomocí stínů různých délek vytvářet iluzi uložení prvků v prostoru a jejich seřazení podle důležitosti.

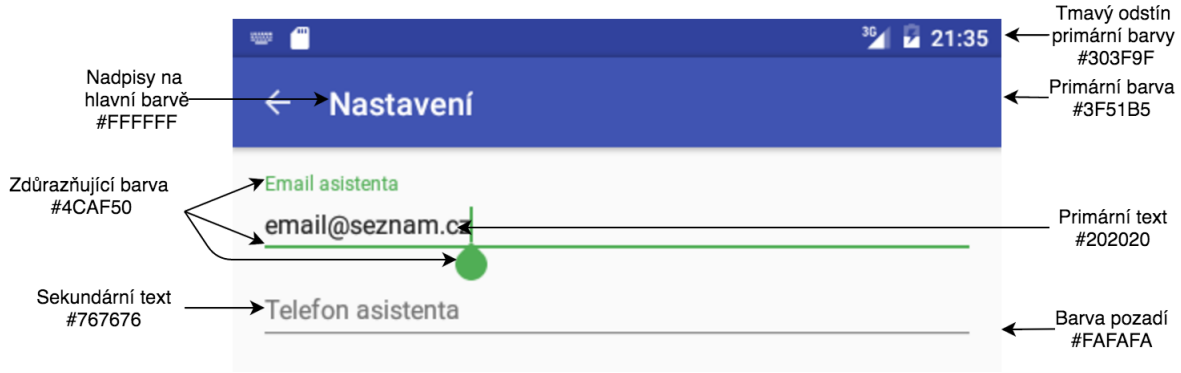
Material design používá ke specifikaci rozměrů stejně jako systém Android jednotku `dp`, což znamená v angličtině „Device independent pixel“ a jedná se o jednotku, která je relativní vůči skutečnému rozlišení zařízení a jejím účelem je zobrazení prvků uživatelského rozhraní v přibližně stejné reálné velikosti bez ohledu na skutečnou hustotu pixelů obrazovky. Obrazovky se zařazují do tříd hustoty a podle této třídy se rozhoduje, kolik skutečných pixelů bude ve výsledku prvek zabírat. Tyto třídy jsou v současnosti `ldpi`, `hdpi`, `tvdpi`, `hdpi`, `xhdpi`, `xxhdpi`

a `xxxhdpi`. Obdélník o šířce 100dp bude pak například v třídě rozlišení `ldpi` zabírat 75 pixelů (1dp = 0.75px), nicméně na zařízení ve třídě `xxxhdpi` bude mít rozměr 400 pixelů (1dp = 4px). Pro uživatelské prvky naší aplikace budeme vždy používat jednotku `dp`.

Specifikace Material designu zavádí jako jednu ze svých hlavních prvků zavedení prostorové osy z do definice uživatelského rozhraní a nazývá tuto vlastnost uživatelských prvků vyvýšení (elevation) [6]. Tato vlastnost ovlivňuje nejvíce velikost stínu uživatelského prvku. Nejvyšší vyvýšení (24dp) mají modální dialogy, poté postranní menu (16dp) následují další komponenty, mezi nimiž jsou námi využívané menu (8dp), vyjíždějící upozornění „Snackbar“(6dp), případně lišta menu rychlých akcí „App bar“ (4dp) [6]. Material design tak přesně určuje chování pro každou komponentu a precizně tak vývojáře vede ke správné implementaci uživatelských prvků.

3.4.2 Barvy

Material design také poskytuje barvy, které by měly být v naší aplikaci použity jako dominantní v hlavních uživatelských prvcích[12]. Pro naší aplikaci byly vybrány jako hlavní barvy odstíny barvy indiga a zdůrazňující barvou byla zvolena zelená. Jako barvy textu byla vybrána čistě bíla pro nadpisy na hlavních barvách, lehce průhledná černá pro primární text a šedá pro sekundární text. Jako barva pozadí těla byla ponechána výchozí lehce ztmavená bílá barva. Popsané barvy s ukázkou můžeme vidět na obrázku 19.



Obrázek 19: Hlavní barvy použité v aplikaci

3.4.3 Ikony

Pro akce aplikace jsou používány ikony ve stylu Material designu, které jsou vždy ploché a snaží se co nejvýstižněji popsat akci, kterou provádí. Ikony lze snadno nalézt na internetu[13], případně importovat přímo v Android Studiu z oficiálních zdrojů Material designu.

V případech těchto ikon je licence použití Creative Common Attribution 4.0 International License (CC-BY 4.0)[14], což znamená, že můžeme bez problému ikony zdarma použít, musíme

[12] Paletu barev můžeme nalézt na https://www.google.com/design/spec/style/color.html (10.4.2016).
[13] Například na https://materialdesignicons.com/ (10.4.2016)
[14] Popis licence na http://creativecommons.org/licenses/by/4.0/ (10.4.2016)

pouze někde v aplikaci uvést, že tyto ikony používáme a v případě, že bychom si je upravili toto také uvést. Příklady používaných ikon můžeme vidět na obrázcích 20, 21 a 22.

Obrázek 20: Nahrávání cesty

Obrázek 21: Přidání fotky

Obrázek 22: Nahrání zvuku

Obrázky je od verze Androidu 5 možné specifikovat ve vektorovém formátu a odpadá tak dříve nutné vytváření mnoha verzí stejného obrázku pro každou třídu rozlišení. Z důvodu zpětné kompatibility pro verze Androidu nižší než 5 je přesto nutné tyto rastrové verze obrázků vytvořit, nicméně vývojové nástroje již tento scénář předpokládají a tyto rastrové verze jsou pro nižší verze systémy vygenerovány během kompilace, což značně zjednodušuje údržbu obrázků.

Pro další ikony loga aplikace a podobně byly použity zdroje ze služby Iconfinder[15], kde je licence pro použití také zdarma. V tomto případě se jedná a Creative Commons Attribution 3.0 Unported (CC BY 3.0) [16], což je starší verze licence popsané výše a v praxi jsou pak podmínky použití ikon stejné.

3.5 Nástroje použité pro návrh

Následující nástroje byly použity pro návrh, úpravy a správu projektu a značně pomohly při jeho vytváření.

3.5.1 Trello

Trello[17] je jednoduchý online nástroj pro správu úkolů, jejich štítkování, zapisování poznámek a přesouvání mezi různými seznamy reprezentujícími kategorii úkolů. V této práci bylo Trello použito pro lepší rozvržení úkolů implementace a úseků psaní práce a to přesto, že na projektu pracoval pouze jeden vývojář. Trello mnoho věcí zpřehlednilo a psaní práce tak byla efektivnější. Logo nástroje můžeme vidět na obrázku 23. Nástroj je přístupný zdarma a nabízí premium edici pro zpřístupnění více funkcí. Pro naše potřeby verze zdarma zcela stačila.

3.5.2 draw.io

Draw.io[18] je online nástroj pro tvorbu grafů, všech různých typů diagramů, myšlenkových map a dalších. Celý editor běží pouze v prohlížeči a synchronizuje vytvářené grafy s připojeným úložištěm Google Drive nebo Dropbox. Grafy jsou tak přístupné a editovatelné odkudkoliv a aplikace je opravdu pokročilá a při práci není vůbec poznat, že vše probíhá pouze v prohlížeči. Umožňuje sdílení i export zhotovených diagramů do mnoha formátů a je tedy velice snadné sdílet

[15] https://www.iconfinder.com/ (10.4.2016)
[16] Popis licence na http://creativecommons.org/licenses/by/3.0/ (10.4.2016)
[17] https://trello.com (10.4.2016)
[18] http://www.draw.io (10.4.2016)

a používat vytvořenou práci. Nástroj byl použit pro vytváření veškerých diagramů a schémat v této práci.

Obrázek 23: Logo nástroje Trello
Zdroj: https://trello.com/home

Obrázek 24: Logo nástroje draw.io
Zdroj: http://www.draw.io

3.5.3 GIMP

GIMP[19] je široce používaný nástroj pro práci s obrázky a jejich úpravu. GIMP znamená *GNU Image Manipulation Program*, což lze volně přeložit jako program pro práci s obrázky s GNU licencí, která nám říká, že program je zdarma a tudíž se hodí pro naše použití. GIMP nabízí obrovské množství funkcí a pro naše účely byl použit zejména pro úpravu snímků obrazovky užitých v této práci.

3.5.4 Android Asset Studio

Specifikem platformy Android je, že spoustu obrázků je potřeba držet v několika verzích pro optimální zobrazení na různých zařízeních, pro zdroje platí určitá pravidla a například poměr velikostí variant obrázků musí splňovat určitá kritéria. Udržovat a vytvářet tyto zdroje je často velmi obtížné a pomoci se nám s tím snaží nástroj Android Asset Studio[20], což je webová aplikace poskytující podporu pro vytváření obrázků v požadovaných velikostech, generování motivů aplikace a podobně. Pro naší aplikaci byl nástroj použit zejména pro generování ikon vyskytujících se v aplikaci a také pro generaci spouštěcí ikony, která podléhá jiným pravidlům než běžné ikony.

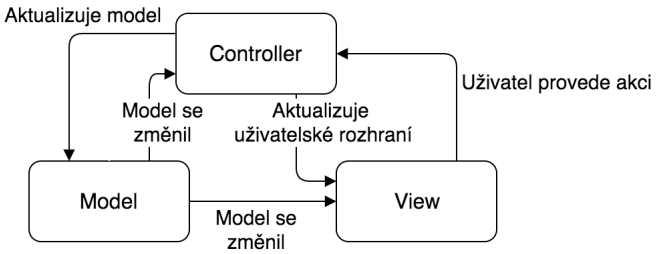
[19] https://www.gimp.org/ (10.4.2016)
[20] https://romannurik.github.io/AndroidAssetStudio/ (10.4.2016)

4 Implementace

Po vytvoření zadání a návrhu bylo třeba tyto myšlenky přeměnit ve funkční aplikaci. V této kapitole si popíšeme, jak byly jednotlivé funkcionality aplikace implementovány a jak byly za tímto účelem použity dříve diskutované nástroje a knihovny. Nejprve si projdeme obecnou architekturu aplikace a následně popíšeme způsob implementace jednotlivých funkcionalit.

4.1 Architektura

Android samotný se snaží uživateli nabízet implementaci Architektury Model View Controller [2], kde každá akce rozhraní view směřuje na controller, který provede příslušnou logiku, případnou komunikaci s modelem - vrstvou poskytující data, kde se dá aktivita považovat jako controller [21], třídy dědící ze systémové třídy `View` která se umí vykreslovat na obrazovku jako view a jako model nabízí systém mechanizmus poskytovatel obsahu (`ContentProvider`), který umožňuje vytvořit v podstatě REST API nad SQLite databází. Zejména pro část modelu se však tato část často implementuje zcela jinak, neboť uživatel nemá tímto způsobem přímý přístup k databázi a poskytovatelé obsahu jsou tak často vhodní více pro komunikaci mezi aplikacemi, kterou přístup k datům založený na URL poskytovatele obsahu umožňuje. Ohledně rozdělení na view a controller je opět v platformě několik nejasností z důvodu nekonzistence API mezi verzemi, kdy ve verzi Androidu 3.0 přibyly jako části aktivity fragmenty a u mnoha vestavěných tříd systému pro View vrstvu (například ListAdapter) provádí i sám systém logiku patřící mimo View. Tento vzor není tedy striktně dodržováno a vzhledem k nutnosti provazovat logiku načítání dat s životními cykly aktivit, které jsou zase svázané s uživatelským rozhraním nelze tedy komponenty zcela precizně oddělit [21]. Schéma architektury MVC můžeme vidět na obrázku 25.

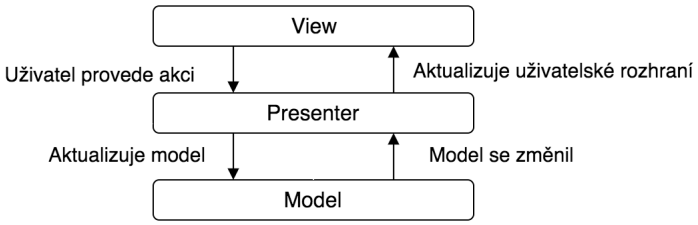


Obrázek 25: Schéma architektury MVC

Díky specifikům platformy existuje mnoho odlišných přístupů, které se snaží platformě Android přizpůsobit tradiční architektury jako již zmíněný Model View Controller, případně podobný Model View, View Model nebo Model View Presenter (MVP) a existuje mnoho projektů, která se snaží ukázat adaptaci těchto modelů pro Android.

Pro náš projekt jsme vybrali architekturu Model View Presenter, která nám lépe umožňuje úplné oddělení modelu od view a všechny aktualizace uživatelského rozhraní se provádí skrz presenter, což je v našem případě aktivita. MVP tak více připomíná vrstvenou architekturu,

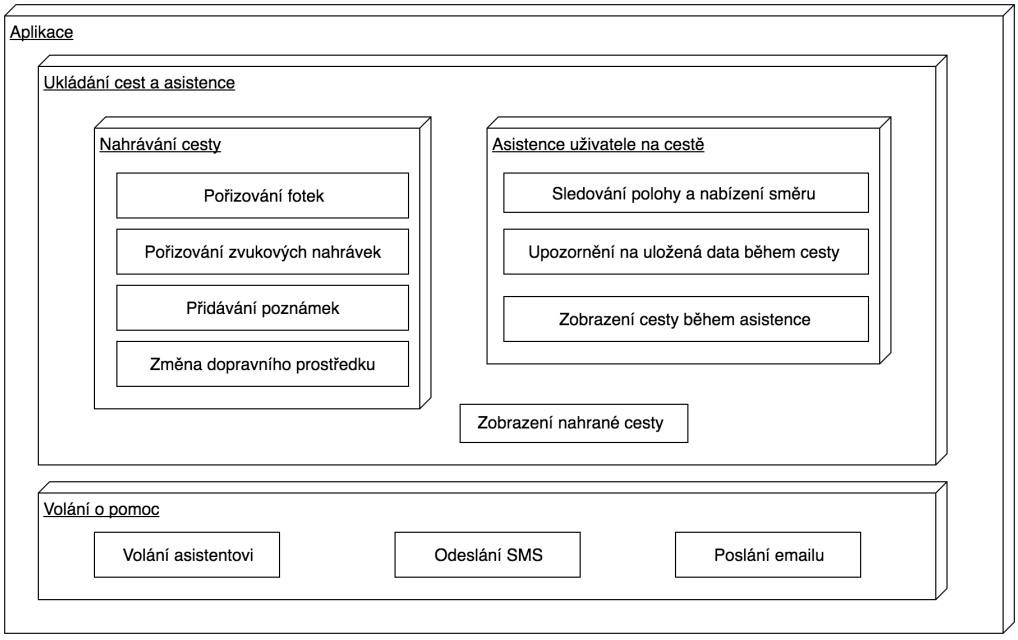
což nám vyhovuje, neboť za vrstvou modelu mohou být další vrstvy komunikace s databází, případně po síti.



Obrázek 26: Schéma architektury MVP použité v projektu

Architektura projektu byla tedy volena s přikloněním k MVP, ve velké míře se používají aktuální úspěšné knihovny a byl kladen důraz na následování známých principů SOLID, popsaných například v [17] a IoC (Inversion of Control), popsaných v [1] spolu s vkládáním závislostí pomocí knihovny Dagger 2, kterou popisujeme v sekci 2.3.4. V projektu je na mnoha místech použito jednotkové testování pro podporu menší chybovosti v kódu za pomocí nástroje Robolectric popsaného v sekci 2.3.9. Přístup k datům a tedy vrstva modelu s propagací výsledků do vyšších vrstev je realizována pomocí RxAndroid (sekce 2.3.2), kde se poskytuje přístup presenterům k těmto datům pomocí objektů `Observable`. Presenter, tedy aktivita, pak publikuje tato data do vrstvy view, kde objekty view zobrazí příslušná data. Fragment v našem případě reprezentuje část view aktivity a je určen například k zobrazení nahrané cesty a všech informací, které k ní byly přidány.

Dílčí funkce aplikace jsou následně popsány v modulech, jakými jsou implementovány a graficky můžeme tyto moduly vidět na obrázku 27.



Obrázek 27: Schéma hlavních funkcí aplikace

4.2 Nahrávání cesty

Během nahrávání cesty se ze systému získávají GPS souřadnice bodů, kterými uživatel se zařízením prochází a k nim se přidávají další data jako fotografie, zvukové nahrávky, poznámky, případně informace o změně dopravního prostředku. Tyto data mají vždy přiřazenou zaznamenanou GPS pozici, která později určí, kdy bude klient na tato data upozorněn.

Při spuštění nahrávání se spustí služba pro nahrávání pozice, která zaregistruje posluchače na systémový objekt třídy `LocationManager` [4], poskytující GPS souřadnice a při obdržení těchto souřadnic vyšleme po aplikační sběrnici událost o nové pozici. Na této sběrnici jsou v tuto chvíli registrování všichni, kteří tuto informaci potřebují znát, což je v tuto chvíli uživatelské rozhraní zobrazující mapu a služba ukládající GPS souřadnice do seznamu, který je v případě, že se uživatel rozhodne cestu uložit uložen do databáze. Jako systémová sběrnice se používá knihovna EventBus a pro možnost registrovat posluchače na poskytovatele GPS souřadnic potřebuje naše aplikace povolení získávat lokaci zařízení.

4.2.1 Pořizování fotek

Během nahrávání cesty může uživatel pořizovat svým telefonem fotky, které by klientovi při asistenci cestování měly pomoci v orientaci a připomenout mu důležité body cesty. Při kliknutí na ikonku přidat fotografii se vytvoří záměr (`Intent`) pořídit fotografii s místem požadovaného uložení voláním metody `startActivity(Intent)` systém spustí výchozí aplikace fotoaparátu na zařízení a o pořízení aplikace se nestará naše aplikace, nýbrž aplikace, na kterou je uživatel zvyklý. Pokud uživatel pořídí fotku a rozhodne se jí uložit, naše aplikace o tom obdrží zprávu, zobrazí se dialog, kde může uživatel fotografii pojmenovat. Vytvoří se unikátní identifikátor fotografie a příslušný záznam asociovaný s nahrávanou cestou. Do záznamu se zapíše aktuální pozice a v případě uložení cesty se tento záznam uloží spolu s cestou.

Jelikož fotografie mohou zabírat poměrně velké množství paměti, vysílá se v případě mazání cesty po systémové sběrnici událost o mazání cesty, na kterou čeká objekt promazávající všechny fotografie a zvukové nahrávky, asociované s touto cestou.

4.2.2 Pořizování zvukových nahrávek

Při nahrávání může uživatel pomocí mikrofonu svého zařízení pořídit zvukovou nahrávku, které se bude vztahovat k aktuálnímu místu. Klientovi tak může poskytnout zvukovou informaci hlasem, který je mu známý a sdělit mu obsáhlejší informace, které by bylo zdlouhavé číst. Pokud uživatel klepne na tlačítko nahrávaní zvuku, spustí se obrazovka ukazující probíhající nahrávání a začne se pomocí systémové služby `MediaRecorder` [4] zaznamenávat zvuk, dokud uživatel neklepne na tlačítko zastavit nebo nezmáčkne tlačítko zpět. V okamžiku otevření nahrávací obrazovky se vytvoří soubor, do kterého systémová služba pro nahrávání začne zapisovat zvuk ve formátu 3GP. Pokud uživatel ukončí nahrávání, může nahrávku ve zobrazeném dialogovém okně pojmenovat a uložit. Nahrávce se následně přiřadí unikátní identifikátor asociovaný s uloženým

souborem a nahrávané cestě se přiřadí záznam přiřazující tuto nahrávku dané cestě. V případě, že uživatel zvolí neukládat nahrávku, nahraný soubor se smaže. Podobně jako v případě fotek jsou při smazání cesty vyčištěny i nahrávky pro zamezení plýtvání místem na úložišti.

4.2.3 Přidávání poznámek

Kdykoliv během nahrávání může uživatel přidat textovou poznámku spojenou s aktuální polohou. Poznámka se zadává do dialogového okna a je poté uložena se zaznamenanou GPS pozicí, ve které byla napsána.

4.2.4 Změna dopravního prostředku

Zvláště se dá uložit změna způsobu transportu uživatele a to klepnutím na tlačítko zobrazující aktuální dopravní prostředek. Pokud například uživatel jde na zastávku, má zobrazenou ikonu chodce, pokud nastoupí do tramvaje a klepne na ikonu chodce, zobrazí se mu dialog s nabídkou pro změnu dopravního prostředku a klepnutím na ikonu tramvaje se tato změna uloží opět s vazbou na aktuální GPS polohu. Uživatel může k této informaci přidat také textový popis například s číslem tramvaje nebo jiné instrukce. Tyto informace se uloží jako záznam do databáze s vazbou na nahrávanou cestu v místě, kde byl záznam pořízen. Ikony používané pro vyjádření druhů dopravních prostředků můžeme vidět na obrázcích 28, 29 a 30.

Obrázek 28: Ikona chůze

Obrázek 29: Ikona autobusu

Obrázek 30: Ikona tramvaje

Změna dopravního prostředku je také místo pro budoucí vylepšení, neboť je možné pomocí akcelerometru zařízení a rozpoznávání statistických vlastností hodnot zaznamenaných akcelerometrem určit tuto informaci automaticky bez účasti uživatele. Knihovna Google Play Services také podporuje API, které se pokouší poskytnout uživateli tuto informaci.

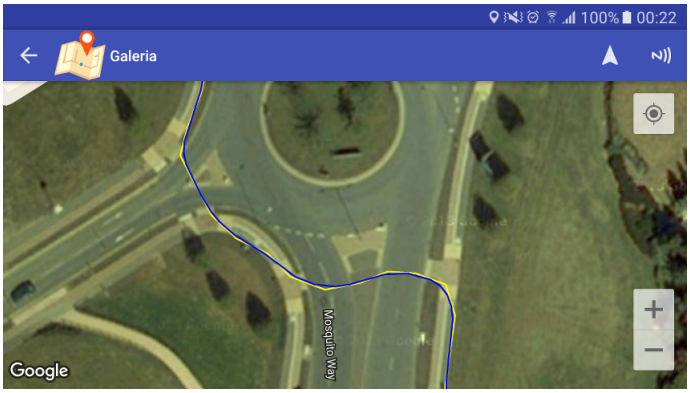
4.3 Zobrazení nahrané cesty

Nahranou cestu je možné na zařízení zobrazit a umožnit tak kontrolu a prohlížení dat. Pro zobrazení slouží fragment mapy ve kterém se zobrazí spojená čára propojující jednotlivé body. Vzhledem k nepřesnostem a šumu při nahrávání však často vzniká velmi lomená čára a dojem z vykreslení na mapě není dobrý. Proto bylo implementováno vyhlazování čar pomocí prokládání aproximační B-spline křivkou, algoritmem inspirovaným v [13]. B-spline křivka je počítána podle rovnice 1, kde je výsledná křivka vytvořena vždy podle 4 sousedních bodů P_0, P_1, P_2 a P_3.

$$B(t) = \frac{(-P_0 + 3P_1 - 3P_2 + P_3)t^3}{6} + \frac{(P_0 - 2P_1 + P_2)t^2}{2} + \frac{(-P_0 + P_2)t}{2} + \frac{(P_0 + 4P_1 + P_2)}{6} \quad (1)$$

Zdroj v [13]

Rovnice 1 se aplikuje zvlášť na GPS souřadnice zeměpisné šířky a délky s krokem $dt = 0.2$, což znamená, že ke každému bodu nám vznikne místo jednoho bodu pět aproximačních bodů a výsledná vykreslená čára je poté hladší, jak můžeme vidět na obrázku 31, kde je žlutou barvou vykreslena čára hrubých dat, uložených u nahrané cesty. Modrou barvou je vykreslena čára s aplikovanou B-spline aproximací. Jak můžeme vidět, aproximace vyhladila ostré okraje zejména u změn směru.



Obrázek 31: Vyhlazení křivky cesty pomocí B-spline aproximace

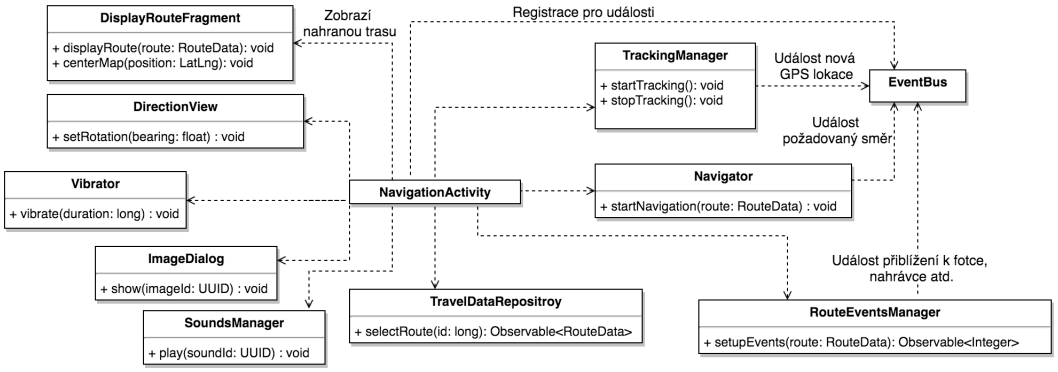
Na mapě jsou dále zobrazeny všechny další data nahraná při cestě. Fotky se nahrají jako miniatury do paměti, oříznou se a zobrazí se jako kruhový výřez nad místem, kde byla fotka pořízena. Velikost miniatury je zadána jako 48dp pro zachování přibližně stejné velikosti na všech zařízeních. Obrázky jsou nahrávány asynchronně pomocí knihovny Universal Image Loader a kruhové oříznutí se realizuje pomocí implementovaného pre-processoru knihovny, která nahranou bitmapu ořízne do kruhového tvaru. Při kliknutí na miniaturu fotky se opět asynchronně načte obrázek do dialogového okna, které zaplní celý displej zařízení s poloprůhledným pruhem, ve kterém je napsán uložený titulek fotky.

Pro zvukové nahrávky a poznámky se zobrazí jednoduchá značka na mapě v místě, kde byla pořízena a při klepnutí na značku se pouze v malém dialogovém okně zobrazí popis a v případě přítomnosti nahrávky se načte z úložiště soubor nahrávky a spustí se. Pro změny cestování se v době zobrazování cesty do mapy zobrazí z výčtu miniatury ikon, příslušných k zaznamenaným datům a změně dopravního prostředku.

4.4 Asistence uživatele na cestě

Pokud uživatel spustí asistenci na cestě, spustí se aktivita navigace, která načte z uložených dat všechny informace, spustí GPS pro obdržení aktuální polohy a připraví vše, co je pro asistenci potřebné. Navádění uživatele je pravděpodobně nejkomplexnější proces aplikace, jehož hlavním řídícím prvkem je třída `NavigationActivity`, která pomocí svých závislostí řeší několik základních úloh, jako je načtení a zobrazování cesty s aktuální polohou uživateli, nabízení následujícího směru, přípravu událostí při přiblížení k místu, kde byl pořízen záznam a upozornění

uživatele v případě přiblížení k těmto místům. Zjednodušený třídní diagram se závislostmi třídy `NavigationActivity` můžeme vidět na obrázku 32, který bude následně blíže popsán.



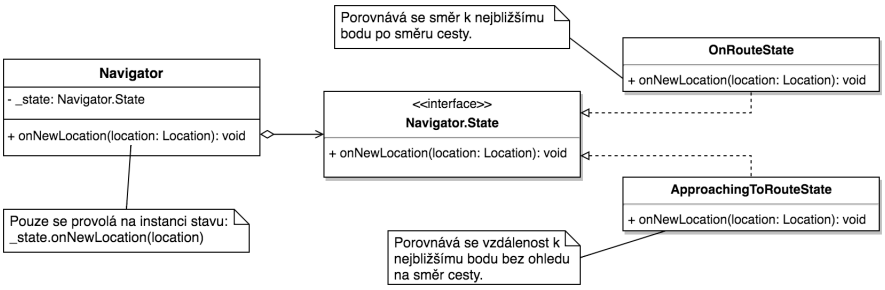
Obrázek 32: Diagram závislostí řídící třídy asistence `NavigationActivity`

Řídící třída `NavigationActivity` dostává při svém spuštění jako parametr identifikátor cesty, pro níž má asistenci spustit. Pomocí třídy `TravelDataRepository` načte asynchronně data k této cestě do paměti a provede všechny úkony potřebné pro asistenci.

4.4.1 Sledování polohy a nabízení směru

Třída `NavigationActivity` volá metodu `startTracking` instance třídy `TrackingManager` a ta začne na sběrnici událostí `EventBus` publikovat události o nové poloze, vždy, když jí obdrží od systému. Instance třídy `Navigator` je na této systémové sběrnici registrována po zavolání metody `startNavigation` a reaguje na tyto události vyhledáváním nejbližšího uloženého bodu na nahrané cestě, ke kterému pomocí metody systémové třídy `Location` vypočítá směr (bearing), kterým by se uživatel měl vydat a který následně publikuje zpět do sběrnice jako událost o požadovaném směru. Na tuto událost reaguje `NavigationActivity` zavoláním metody `setRotation` na instanci `View`, obsahující obrázek šipky ukazující daným směrem.

Třída `Navigator` využívá k vyhledávání nejbližšího bodu návrhový vzor Stav (State [3]), jelikož se uživatel při používání může nacházet ve dvou různých stavech. Diagram implementace vzoru můžeme vidět na obrázku 33. První je při počátku navigace, kdy se přibližuje k nahrané cestě (`ApproachingToRouteState`) a při obdržení události o nové GPS pozici se prohledají body cesty a najde se nejbližší, ke kterému se spočítá vzdálenost a pokud je vzdálenost menší, než zadaná hranice, vyhodnotí se, že je uživatel již na cestě a stav se přepne na `OnRouteState`. V tomto stavu se při obdržení nové GPS pozice spočítá podle směru cesty bod, který je nejblíže aktuální pozici, nicméně leží ve směru cestování. Opět se počítá vzdálenost k cestě a pokud překročí určitou hranici, přepne se `Navigator` zpět do stavu `ApproachingToRouteState` a začne uživatele vést zpět přímo k cestě.



Obrázek 33: Využití návrhového vzoru stav ve třídě `Navigator`

4.4.2 Zobrazení cesty během asistence

Uživateli se podobně jako při prohlížení detailu cesty zobrazí mapa s vyznačenou nahranou cestou. Pro zobrazení těchto dat slouží fragment `RouteDisplayFragment`, který aplikuje vyhlazování cesty a zobrazí ji na mapě, která se s pohybem uživatele centruje podle jeho GPS pozice, jelikož `NavigationActivity` během pohybu obdrží událost o nové pozici a zavolá metodu fragmentu `centerMapTo`.

4.4.3 Upozornění na uložená data během cesty

Během spuštění asistence pro cestu se pomocí třídy `RouteEventsManager` připraví všechny události, které se budou během cestování spouštět. Využívá se pro to Geofencing API[21] z knihovny Google Play Services, kde pomocí API knihovny připravíme události, které instalovaná knihovna na zařízení sama vyvolá, jakmile se přiblížíme k zadanému místu. V naší aplikaci se události knihovny při přiblížení k místům záznamů přeloží na události aplikace a přes systémovou sběrnici se publikují dále. Na tyto události čeká opět třída `NavigationActivity`, která při obdržení události pomocí systémové služby `Vibrator` zavibruje zařízením, v případě fotky zobrazí pomocí třídy `ImageDialog` dialogové okno s fotkou, u poznámky dialogové okno s textem a u zvukové nahrávky nahrávku přehraje třída `SoundsManager`, která podle identifikátoru vyhledá nahrávku v úložišti a spustí ji.

4.5 Volání o pomoc

Během cestování se klient může dostat do různých situací a naše aplikace by mu měla umožnit rychle se spojit se svým asistentem pro vyřešení problému, případně mu poslat zprávu. Asistent může v aplikaci nastavit své kontaktní údaje a aplikace může následně tyto údaje využít k rychlému kontaktování. Klepnutím ikony kontaktovat asistenta se klientovi zobrazí tlačítka s možnostmi.

[21] Dokumentace na http://developer.android.com/training/location/geofencing.html

4.5.1 Volání asistentovi

Pokud má uživatel v aplikaci uloženo číslo asistenta, po klepnutí na možnost volat se v kódu vytvoří objekt záměru (`Intent`) se specifikací, že se na toto číslo má vytočit hovor. Systém vyhledá aplikaci, která dokáže tento záměr vykonat a spustí vytáčení. Hovor je v tuto chvíli zpoplatněn jako jakýkoliv jiný hovor z telefonu uživatele.

4.5.2 Odeslání SMS

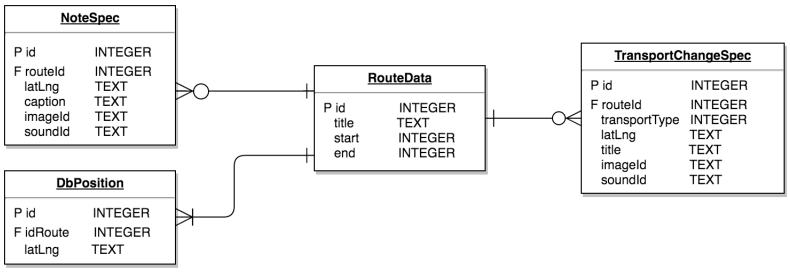
Pokud klient zvolí tuto možnost, vytvoří se SMS zpráva s textem, obsahující odkaz na Google mapy se zakódovaným ukazatelem na místo, kde se klient nacházel v odesílání zprávy. Asistent si tak po obdržení SMS zprávy může snadno zobrazit, kde se uživatel, který má problémy nachází. Při vytváření a posílání zprávy dbá aplikace na to, aby zpráva neobsahovala více než 160 znaků a klient tak zaplatil pouze za jednu SMS. Aplikace využívá pro odeslání SMS systémové služby `SmsManager` [4].

4.5.3 Poslání emailu

Podobně jako při odesílání SMS je připravena zpráva s odkazem na Google mapy, která opět asistentovi umožní rychlý přístup k aktuální pozici uživatele. Vytvoří se objekt záměr pro odeslání emailu s předvyplněnou adresou asistenta, pokud je tato adresa v nastavení a voláním metody `startActivity(emailIntent)` spustí systém emailovou aplikaci na zařízení pro odeslání emailu z klientova účtu. Spouštění emailové aplikace na zařízení nám umožňuje nepožadovat od uživatele jeho zabezpečovací údaje k emailové schránce, což by bylo pro uživatele velmi nepříjemné.

4.6 Ukládaní dat

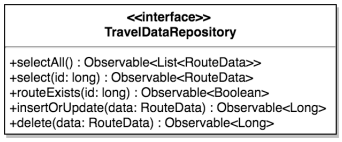
V naší aplikaci je potřeba ukládat zaznamenané souřadnice cesty, poznámky, fotky, změny dopravního prostředku, zvukové nahrávky a další pomocná data, jako názvy fotek a časy nahrání cesty. Pro tyto účely byla zvolena SQLite databáze, pro kterou Android poskytuje API, v kombinaci se souborovými úložišti pro zvukové nahrávky a fotografie. Do databáze se uloží hlavní záznam nahrané cesty do tabulky `RouteData` s časy nahrání a názvem cesty, tabulka `DbPosition` obsahuje nahrané GPS souřadnice s vazbou na hlavní záznam cesty, tabulka `NoteSpec` poznámky ke konkrétnímu místu s případnými identifikátory souborů fotek nebo nahrávek a tabulka `TransportChangeSpec` informace o změně dopravního prostředku. ER diagram databázového schématu můžeme vidět na obrázku 34. SQLite používá místo typických datových typů koncept afinit [20], kterých je 5 a to `INTEGER`, `TEXT`, `BLOB`, `REAL` a `NUMERIC`. Pro naše účely bylo potřeba pouze využít prvních dvou, neboť typ `INTEGER` může podle potřeby mít až 8 bytů [20], což nám bez problému umožňuje do něj uložit beze ztráty hodnoty typu `long` jazyka Java, kterou používáme pro naše hodnoty id.



Obrázek 34: ER diagram databázového schéma

4.6.1 Přístup k databázovým datům

Pro práci s databází byla využita knihovna DBFlow, popsaná v sekci 2.3.8, díky které jsme pro vytvoření požadované databázové struktury pouze vytvořili třídy pro mapování, které jsme anotovali příslušnými anotacemi knihovny. Ukládání dat pak probíhá přes tyto objekty a k nim vygenerované adaptéry. Pro přístup k datům byl použit vzor Repozitář [12] (Repository), díky kterému jsou všechny operace s daty prováděny přes jednu vrstvu a oddělují tak kód práce s daty od zbytku aplikace jedním rozhraním, za kterým se může nacházet námi implementovaná databázová vrstva, ale také lze toto pak snadno zaměnit za webovou službu, případně pouze čtení ze souboru. Bylo vytvořeno rozhraní `TravelDataRepository`, které můžeme vidět na obrázku 35, pro které existuje implementace pomocí DBFlow a objekt této implementace je pomocí vkládání závislostí vložen kdekoliv do aplikace, kde si o to některé třída zažádá, jelikož potřebuje přístup k datům. Rozhraní vrací všechny hodnoty jako `Observabe` a tím nechává na klientovi, aby se rozhodl, zda chce data načíst synchronně, asynchronně, případně umožňuje provést další operace.



Obrázek 35: Rozhraní pro přístup k datům aplikace

4.6.2 Přístup k souborovým datům

Jelikož aplikace vytváří i souborová data, je potřeba tato data ukládat a provázat je s daty v databázi. Pro naše dva případy fotek a zvukových nahrávek u poznámek bylo použito vygenerování náhodného objektu `UUID` při vytváření souboru, soubor se následně v příslušném adresáři pro tento typ dat podle tohoto identifikátoru pojmenuje a hodnota identifikátoru se uloží do asociovaného sloupce v databázi. Při načítání dat se tak aplikace, například pro obrázky, pouze podívá, zda záznam obsahuje identifikátor obrázku a pomocí získaného identifikátoru získá z manažera obrázků přístup ke konkrétnímu souboru, který poté může zobrazit uživateli.

4.6.3 Ukládání nastavení

Pro uživatelská nastavení, jako například číslo asistenta nebo jeho email, je použit mechanismus Androidu `SharedPreferences`, který v našem případě pouze obalujeme naším objektem s přístupovými metodami pro konkrétní, námi používaná nastavení.

4.7 Zálohování dat

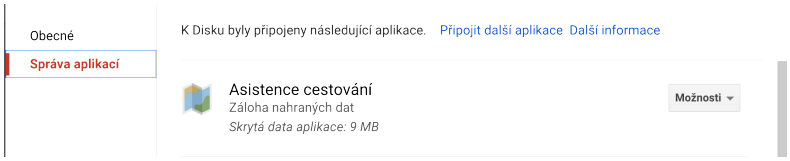
Jelikož asistent může strávit poměrně dost času sběrem dat a jejich přípravou pro klienta, je žádoucí chránit data před ztrátou, poškozením nebo pouze nechtěným smazáním či smazáním odinstalováním aplikace. Aplikace tudíž obsahuje mechanismus pro zálohu všech dat a to nahraných cest s poznámkami, upozorněními na změnu cesty, fotkami a nahrávkami.

Pro fungující zálohování je nutné přiřadit uživateli identitu a tu rozpoznávat a ověřovat pro přiřazení případné zálohy, mít k dispozici webové úložiště, komunikovat s ním a porovnávat informace z něj získané s aktuálními daty, což je poměrně komplexní úkol a vlastní implementace by vyžadovala vlastní server, hosting a zejména registraci uživatele, což by v případě mentálně postižených klientů přineslo další složitost do aplikace a nutilo by klienta pamatovat si další uživatelské jméno a heslo.

Tento problém byl vyřešen využitím OAuth2[22] autorizace přes připojený Google účet na zařízení a API Google Drive nebo česky Disk, které je součástí Google Play Services a na zařízení uživatele, který si naši aplikací stáhnul z obchodu Play, jsou tato API k dispozici. V Google vývojářské konzoli je napřed nutné aktivovat využití Google Drive API pomocí jména balíčku aplikace a SHA-1 heše certifikátu, kterým je podepsáno APK aplikace. Na stejném místě lze také nastavit popis, který se pro aplikaci uživateli zobrazí, když si prohlíží, které aplikace má ke svému Disku připojeny. Díky API Google Drive je odpovědnost synchronizace dat přesunuta na služby Googlu a pomocí OAuth2 se využívá existující účet, čímž se odstraňuje nutnost uživatele zakládat a pamatovat další identitu a nám zcela eliminuje nutnost starat se o synchronizaci mezi zařízeními a správu jakýchkoliv účtů. Uživatel musí při prvním vytvoření zálohy potvrdit, že souhlasí, aby naše aplikace používala jeho Disk. API s tímto počítá a při pokusu o vytváření souborů při nutnosti autorizace sám nabídne ke zobrazení dialog pro uživatele a následně vrátí výsledek, zda uživatel službu povolil.

Data záloh jsou ukládána do tzv. appfolderu [5], což je privátní úložiště aplikace a uživatel poté v nastavení svého Disku vidí, která aplikace tuto službu využívá a kolik dat zabírá, nevidí však konkrétní data. Uživatelův pohled po připojení aplikace a zálohování dat můžeme vidět na obrázku 36.

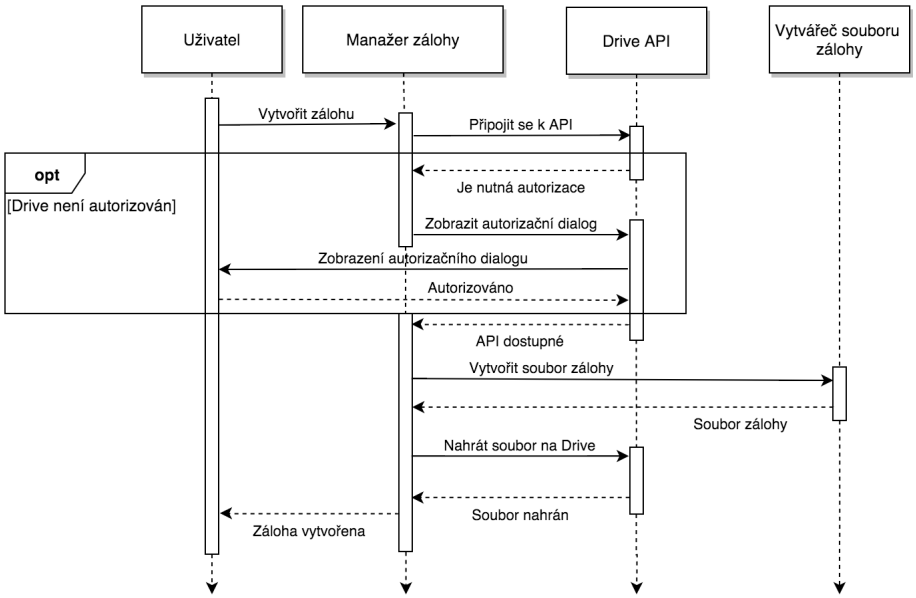
[22] Více o OAuth2 se můžeme dozvědět v [9].



Obrázek 36: Uložená data aplikace na Google Drive
Zdroj: https://drive.google.com/drive/my-drive

4.7.1 Záloha

Pro vytvoření zálohy bylo za účelem minimalizace závislosti na Google Drive API zvoleno vytvoření jediného dočasného ZIP souboru s databází a soubory, který se následně uloží na Disk uživatele. Dočasný soubor se po vytvoření zálohy smaže a stejně tak předchozí zálohy na disku, pro omezení využívání příliš mnoha místa duplicitami, jelikož díky fotkám může být zálohový soubor poměrně velký. Diagram vytváření zálohy můžeme vidět na obrázku 37.



Obrázek 37: Diagram vytváření zálohy

4.7.2 Obnovení ze zálohy

Pokud uživatel požádá o obnovení ze zálohy, vyhledá se pomocí Drive API poslední ZIP soubor zálohy, rozbalí se do dočasného úložiště, soubory se přesunou do příslušných složek, uzavře se existující databázové připojení aplikace, nahradí se databázový soubor souborem ze zálohy a připojení se znovu otevře. Uživatel může následně začít využívat obnovená data.

4.8 NFC (*Near Field Communication*)

NFC je bezdrátová technologie umožňující komunikaci dvou zařízení na velmi krátké vzdálenosti do 4 cm rychlostí do 424 kb/s [15]. Výměna dat může být aktivována pouhým přiblížením dvou zařízení k sobě a technologie podporuje zabezpečený přenos dat. Technologii spravuje organizace NFC Forum jejíž logo lze vidět na obrázku 38 a její specifikaci lze nalézt v [11]. NFC dále implementuje specifikaci ISO/IEC 14443 A a B [15], což je standard používaný současnými platebními kartami a jinými čipy, jehož definici můžeme najít v [18] a díky této implementaci je plně kompatibilní se současnými platebními terminály a systémy, což nabízí možnost využití zařízení podporujících NFC jako platebních čipů jelikož oproti klasickým čipům může NFC zařízení uchovávat více informací a umožňuje například sloučení více platebních karet a bezpečnostních čipů dohromady do jednoho zařízení, v našem případě mobilního telefonu.

Během posledních pár let se NFC stalo součástí výbavy mnoha chytrých telefonů a velká většina modelů už nyní tuto funkčnost podporuje a Android nabízí ve svém API podporu pro NFC už od verze 2.3 vydané v roce 2010 a vývojář je tedy odstíněn od všech nízkoúrovňových komunikačních protokolů. V naší aplikaci využijeme tzv. NFC tagů, což je malý vestavěný čip schopný pomocí NFC uchovat malé množství dat a následně touto technologií tyto data přečíst. Tagy se dají v současnosti koupit na internetu za desítky korun a mohou být různých tvarů a barev.

V aplikaci Asistence cestování bylo implementováno uložení URL s informací o nahrané cestě. Funkce pracující s NFC jsou v aplikaci označeny jednotnou ikonou na obrázku 39, která je nejčastěji používána Android aplikacemi. Pokud uživatel klepne na ikonu NFC při prohlížení detailu cesty, zobrazí se mu obrazovka s instrukcemi a po přiložení zapisovatelného NFC tagu je toto URL uloženo do tagu spolu s informací, že právě naše aplikace umí toto URL rozpoznat. V Android Manifestu je následně definována aktivita s intent filtrem na akci *NDEF_DISCOVERED*, což systému říká, že při objevení NFC tagu má zařadit naši aplikaci na seznam aplikací, které dokážou rozpoznat NFC tagy a spolu s námi zapsanou informací do tagu je to právě naše aplikace, která při přiložení tagu zprávu obdrží, dekóduje URL a podle jeho obsahu spustí navigaci nalezené cesty.

Výsledkem je, že uživateli se při přiložení konkrétního tagu zobrazí připravená navigace pro danou cestu bez jakékoliv další akce, což značně zjednodušuje používání aplikace, jelikož jinak by pro spuštění navigace uživatel musel aplikaci spustit, nalézt požadovanou cestu a klepnout na ikonku navigace. Nápadem využití je připevnění několika NFC tagů rozlišených barvami vedle dveří bytu klienta a při odchodu na cestu uživatel spustí asistenci k cestě přiložením telefonu k požadované barvě.

Obrázek 38: Logo organizace NFC Forum
Zdroj: http://nfc-forum.org/

Obrázek 39: Ikona NFC
Zdroj: https://materialdesignicons.com/

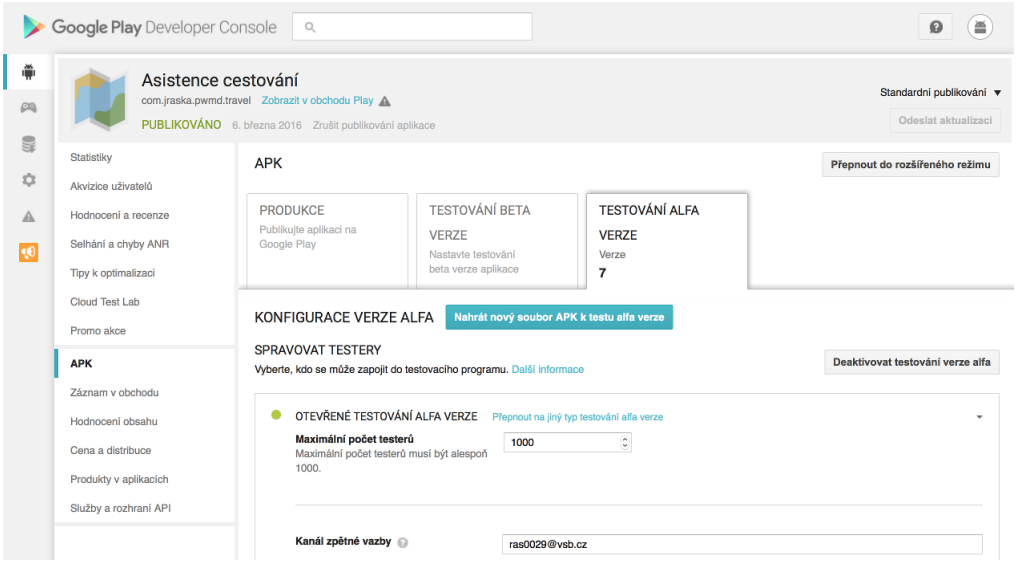
4.9 Publikování aplikace

Pro publikování aplikace byl zvolen obchod Google Play provozovaný Googlem, což je standardní způsob publikování Android aplikací. Google Play poskytuje bohaté rozhraní pro správu aplikací, statistiky instalovaných zařízení, verzí systému používajících naší aplikaci, reporting chyb a další. Poskytuje také nástroje pro propagaci aplikace a v neposlední řadě správu verzí aplikace, grafiky a popisů vztahujících se k aplikaci. Naše aplikace je v tomto obchodě samozřejmě přístupná zdarma, nicméně je možné publikovat placené aplikace, kde Google Play řeší distribuci do všech vybraných států a za služby prodeje si strhává provizi 30%.

Od nedávna obchod podporuje také otevřené alfa a beta testování, kde lze uživatele zvát anonymně pouze za pomoci odkazu. Dříve bylo nutné zakládat vývojářské skupiny a přidávat do nich jmenovitě lidi, což bylo zdlouhavé a pracné. Po odeslání testovacího odkazu se obchod uživatele pouze zeptá, zda se chce stát testerem a může si následně stáhnout aplikaci, která je jinak pro veřejnost nepřístupná Pro původní testování aplikace je využit alfa kanál pouze pro úzký okruh uživatelů, následně následuje beta kanál pro širší testování a odhalení chyb, které stále nejsou mezi všemi uživateli, nicméně uživatelů je dost na to, aby se chyby projevily. Po tomto kroku následuje povýšení aplikace do produkce, kde je již veřejně přístupná a může si ji stáhnout i náhodný uživatel, který aplikaci v obchodě objeví. Ukázku vzhledu publikování aplikace pro obchod Play můžeme vidět na obrázku 40.

Existují také alternativní obchody, ze kterých si lze stáhnout aplikace, z nichž největší je obchod Amazonu. Podíl těchto obchodů je minoritní a pokud uživatel chce tyto alternativní obchody používat, musí na zařízení v nastavení povolit instalaci aplikací z neznámých zdrojů, což je značné bezpečnostní riziko. Dříve často diskutovaná chabá bezpečnost systému byla často zapříčiněna uživateli využívající tyto alternativní obchody, kde neexistovala verifikace uživatele nebo obsahu a mohl na ně prakticky kdokoliv nahrát škodlivou aplikaci zdarma tvářící se jako některá z populárních placených aplikací.

Pro publikování na Google Play je nutné aplikaci podepsat produkčním certifikátem, který byl vygenerován s platností na 50 let a je uložen mimo systém správy verzí, jelikož se jedná o esenciální bezpečnostní prvek, neboť při jeho zcizení by nabyvatel mohl publikovat aplikaci se stejným podpisem jako naše aplikace a distribuovat tak škodlivý kód pod naším jménem.



Obrázek 40: Rozhraní publikování aplikace na Google Play

5 Testování aplikace

Po nasazení aplikace bylo potřeba identifikovat a odladit všechny možné problémy a nedostatky. Aplikace byla pomocí alfa a beta kanálů obchodu Google Play distribuována testovacím uživatelům, kteří se po otevření zaslaného odkazu stali testery. V této kapitole si představíme úpravy spojené s jejich zpětnou vazbou a dojmy testerů.

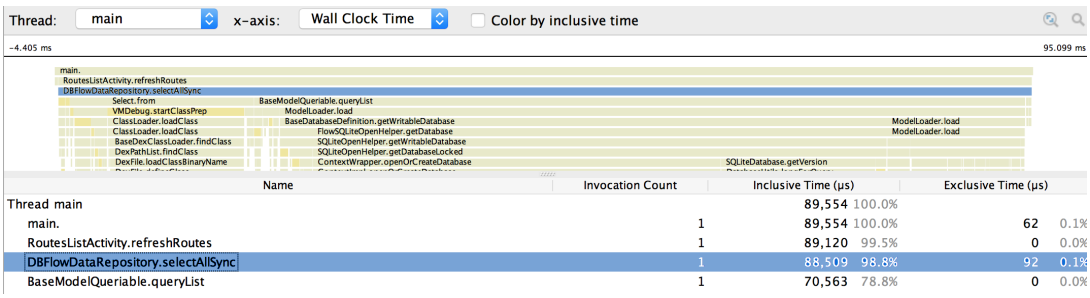
5.1 Optimalizace výkonu

Jedno z pravidel programování je neoptimalizovat zbytečně dopředu na základě odhadu, jelikož programátoři jsou v těchto odhadech většinou velmi špatní. Během vývoje byl přesto na efektivní implementaci kladen důraz a všechny potencionálně delší operace byly prováděny asynchronně mimo vlákno uživatelského rozhraní, jelikož pro optimální uživatelský zážitek je potřeba stihnout vše ve vlákně uživatelského rozhraní do 16ms [19].

Aplikace, která chce toto uživateli umožnit, musí do asynchronních operací přesunout většinu svých déle trvajících operací, což bylo v našem případě zejména čtení a ukládání do databáze, jelikož se v tomto případě komunikuje se souborovým systémem, ale také například využívání API Google Play Services, kde může dojít k ověřování využívání API přes internet což už z principu znamená poměrně dlouhou prodlevu. Pro implementaci těchto operací byla použita knihovna RxAndroid, popsaná v sekci 2.3.2 a aplikace se poté stává mnohem rychlejší a zejména odezva uživatelského rozhraní na akce uživatele je téměř okamžitá.

5.1.1 Pomalé načtení úvodní obrazovky s cestami

Při testování se na pomalých zařízeních projevilo pomalé načítání úvodní obrazovky se seznamem cest. Bylo tedy využito profilovacího nástroje Traceview, který je součástí Android SDK a je možné jej využít pro zaznamenání a zobrazení provádění metod za běhu. Výstup profilování kódu mezi synchronním načtením v metodě `refreshRoutes` a zobrazením výsledku v uživatelském rozhraní můžeme vidět na obrázku 41, kde je výstup hlavního vlákna uživatelského rozhraní. Z důvodu předcházení nedorozumění je nutno poznamenat, že profilovací výstupy jsou lokalizovány v angličtině a tudíž je čárka brána jako oddělovač tisíců, nikoliv jako oddělovač desetinných míst, pro který je používána tečka. 88,509μs ve výstupu tedy znamená 88ms a 509μs.

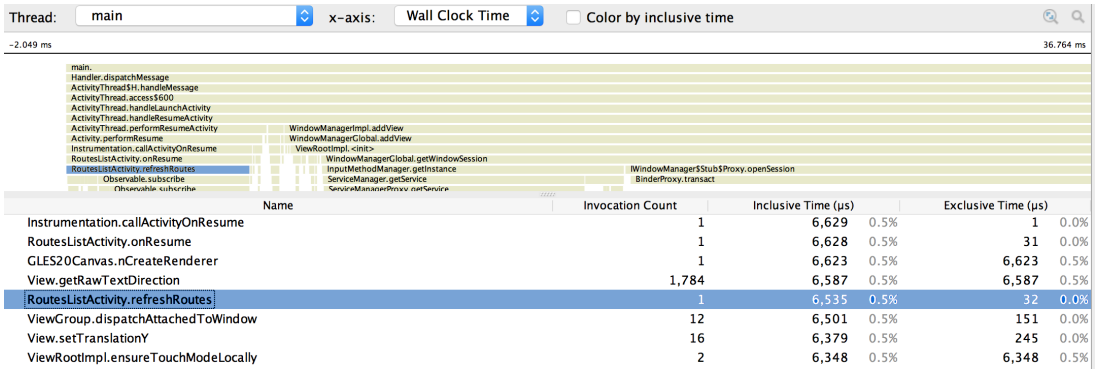


Obrázek 41: Výstup profilování synchronního načítání cest

Jak na obrázku 41 můžeme vidět, metoda `selectRoutesSync` zabírá 98.8% z celého načítání a zobrazení dat a trvá více než 88ms, což sice nemusí být kritické, nicméně se jedná pouze o čas na testovacím zařízení a na některých pomalejších zařízeních s například starým typem SD karty může být tento čas mnohem vyšší a uživateli již může přivodit pocit pomalé odezvy. Na časové ose také můžeme identifikovat důvody pomalého zobrazení při prvním načtení. Při prvním volání se totiž v metodě `ClassLoader.loadClass` načítají do paměti všechny třídy potřebné pro načtení dat z databáze, což jsou v tomto případě naše mapovací třídy reprezentující data v paměti, třídy ORM knihovny DBFlow a také třídy samotného frameworku Android pro přístup k SQLite databázi. Významnou roli hraje také otevírání databáze, které proběhne v metodě `SQLiteOpenHelper.getWritableDatabase` a opět probíhá při prvním přístupu k databázi a samotné načtení dat, probíhající v metodě `ModelLoaer.load` je v tomto případě minoritním konzumentem času vlákna.

Pro optimalizaci bylo tedy využito asynchronního načtení, kde se pomocí využití knihovny RxAndroid přesunulo volání těchto metod do jiného vlákna a následně se do hlavního vlákna publikoval pouze výsledek asynchronní operace. Implementaci takového volání jsme mohli vidět na ukázce kódu 1 na straně 20.

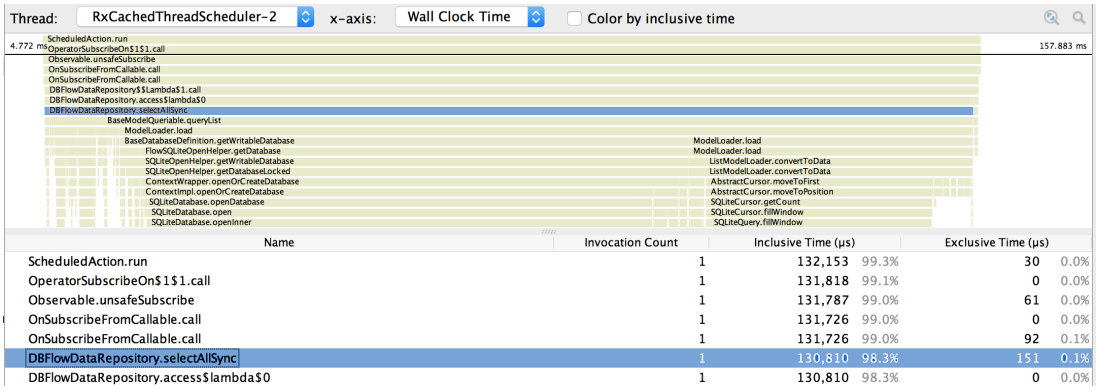
Při opětovném profilování, jehož výstup můžeme vidět na obrázku 42, se nám povedlo významně snížit čas volání metody `refreshRoutes` na mnohem lepších 6.5ms a majoritní část práce hlavního vlákna tvoří vykreslování uživatelského rozhraní v metodě `doFrame`, což je přesně to, čeho jsme chtěli dosáhnout.



Name	Invocation Count	Inclusive Time (µs)		Exclusive Time (µs)	
Instrumentation.callActivityOnResume	1	6,629	0.5%	1	0.0%
RoutesListActivity.onResume	1	6,628	0.5%	31	0.0%
GLES20Canvas.nCreateRenderer	1	6,623	0.5%	6,623	0.5%
View.getRawTextDirection	1,784	6,587	0.5%	6,587	0.5%
RoutesListActivity.refreshRoutes	1	6,535	0.5%	32	0.0%
ViewGroup.dispatchAttachedToWindow	12	6,501	0.5%	151	0.0%
View.setTranslationY	16	6,379	0.5%	245	0.0%
ViewRootImpl.ensureTouchModeLocally	2	6,348	0.5%	6,348	0.5%

Obrázek 42: Výstup profilování hlavního vlákna asynchronního načítání cest

Ze stejného profilovacícho výstupu se lze přepnout do záznamů z asynchronního vlákna, který je na obrázku 43, kde můžeme vidět, že provádění metody `selectRoutesSync` nyní trvá 130ms, nicméně se tak děje v pracovním vlákně aplikace a uživatel tak nezaznamená zaseknutí uživatelského rozhraní při zobrazování obrazovky a po prodlevě načítání se mu pak do připraveného uživatelského rozhraní pouze zobrazí samotné cesty.



Name	Invocation Count	Inclusive Time (µs)		Exclusive Time (µs)	
ScheduledAction.run	1	132,153	99.3%	30	0.0%
OperatorSubscribeOn$1$1.call	1	131,818	99.1%	0	0.0%
Observable.unsafeSubscribe	1	131,787	99.0%	61	0.0%
OnSubscribeFromCallable.call	1	131,726	99.0%	0	0.0%
OnSubscribeFromCallable.call	1	131,726	99.0%	92	0.1%
DBFlowDataRepository.selectAllSync	1	130,810	98.3%	151	0.1%
DBFlowDataRepository.access$lambda$0	1	130,810	98.3%	0	0.0%

Obrázek 43: Výstup profilování pracovního vlákna asynchronního načítání cest

Pro pomalé zařízení byl tedy problém se zablokováním uživatelského rozhraní takto vyřešen. Podobné profilování bylo provedeno preventivně i u ostatních obrazovek, neboť výstup profilovací nástroje může velmi často odhalit nejen výkonnostní problémy, ale také chyby. Žádné zásadní problémy nebyly v době psaní této práce objeveny.

5.1.2 Průběžné monitorování výkonu

Jak již bylo řečeno, není nevyžádaná optimalizace doporučeným postupem, jelikož je optimalizovaný kód často složitější a hůře čitelný. Během vývoje byl proto využit nástroj Hugo[23], který pouhým anotováním metody anotací `@DebugLog` začne do logu zařízení vypisovat informace o vstupech metody, jejím výstupu a také čas provádění. Činí tak pouze v ladícím režimu, což je to, co od něj požadujeme a aplikace používaná finálními uživateli tedy není tímto logováním nijak ovlivněna. V projektu byly takto anotovány všechny potencionálně pomalé metody a po vyfiltrování těchto metod ve výstupu logů se tak lze snadno podívat, které metody opravdu zabírají více času a umožňuje nám ujistit se, že jsou tyto metody spouštěny v asynchronním vlákně. Ukázku použití můžeme vidět na ukázce kódu 6, kde zjistíme, že načtení detailu cesty trvá 248ms, což by již bylo velmi problematické provádět v uživatelském vlákně, nicméně k naší spokojenosti také vidíme, že se toto provádí v asynchronním vlákně knihovny RxAndroid.

```
@DebugLog
private RouteData selectSync(long id) { ... }
        ↓                ↓
V/DBFlowDataRepository: selectSync(id=4) [Thread:"RxCachedThreadScheduler-1"]
V/DBFlowDataRepository: selectSync [248ms] = RouteData(_deletedId=0, _id=4, _start...
```

Výpis 6: Využití nástroje Hugo pro výpisy informací o prováděných metodách

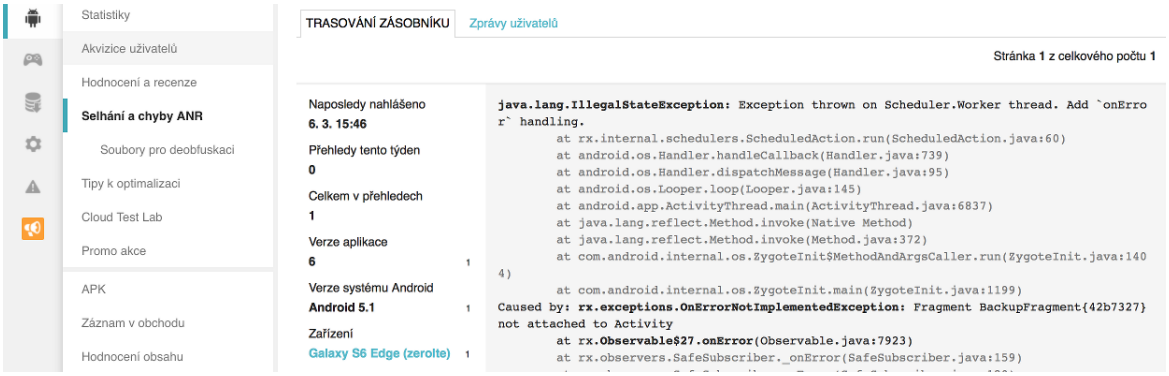
[23] Dostupné z https://github.com/JakeWharton/hugo (10.4.2016)

5.2 Pády aplikace

Jedním z nejnepříjemnějších problémů, co může uživatele potkat, je pád aplikace a neslavné dialogové okno s dotazem, zda chceme aplikaci ukončit. Bývá poté velmi těžké uživatele přesvědčit, aby se do aplikace vrátil. Při každém vydávání verze byly proto vždy otestovány všechny hlavní scénáře, kterými aplikace prochází, nicméně i přesto se můžou chyby dostat do produkce a je důležité se o nich dozvědět.

Pro tyto účely se opět používají služby obchodu Google Play a knihovny Google Play Services instalované na zařízení. Pokud nastane za běhu aplikace výjimka a aplikace spadne. Zobrazí se uživateli nabídka k odeslání zpětné vazby spolu s informacemi o chybě. testovací uživatelé byli instruováni toto udělat a vzhledem k dřívějším testům byla takto nahlášena pouze jedna výjimka.

Nahlášené chyby si lze poté prohlédnou v konzoli obchodu Google Play, kde se nahrají také další důležité informace o zařízení, jako verze systému, výrobce, typ zařízení a zejména trasování zásobníku chyby, podle kterého je snadno možné chybu odhalit. Ukázku nahlášené chyby můžeme vidět na obrázku 44.



Obrázek 44: Nahlášený pád aplikace v konzoli Google Play

Při procházení výpisu bylo zjištěno, že chyba je způsobena voláním metod fragmentu pro získání textových zdrojů vyžadujících připojenou aktivitu, nicméně volání této metody přišlo v okamžiku, kdy aktivita rotovala a tudíž byl fragment již z aktivity odstraněn, což vedlo k nahlášenému selhání.

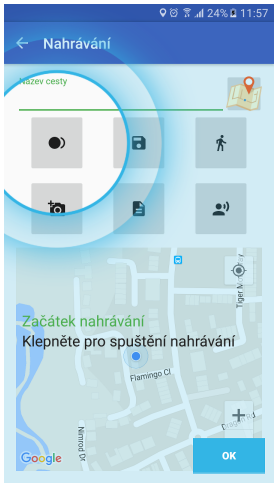
Chyba byla vyřešena kontrolou stavu a získáváním textových zdrojů již při vytváření fragmentu, kdy máme jistotu, že je fragment k aktivitě správně připojen. Po nahrání opravné verze na Google Play byla nové verze distribuována během pár hodin, byla tak opravena a jak můžeme vidět na obrázku 44, objevila se pouze jednou.

5.3 Napovídání uživateli

Jak se během testování ukázalo, někteří uživatelé si při prvním spuštění obrazovky nahrávání nebyli jistí, čím mají začít a přesto, že jediné tlačítko, na které je v tu chvíli možné klepnout

je tlačítko nahrávání s ikonou, kterou jsme mohli vidět na obrázku 4 a při podržení prstu na tlačítku s touto ikonou se zobrazí popis akce.

Bylo proto potřeba při prvním spuštění uživateli ještě výrazněji ukázat, kde má na této obrazovce začít. Pro tento účel bylo použito překrytí celé obrazovky poloprůhlednou vrstvou, na které je napsán text, který radí uživateli, že má klepnout na tlačítko nahrávání a toto tlačítko se nachází v kruhové oblasti nápovědy, která je jako jediná v ten okamžik průhledná. Uživateli se toto zobrazí pouze pokud obrazovku otevře poprvé a upozornění se skryje, pokud provede jakoukoliv uživatelskou akci a informace o tom, že nápověda byla uživateli již zobrazena se uloží do nastavení zařízení. Zobrazení nápovědy můžete vidět na obrázku 45.



Obrázek 45: Zobrazení nápovědy uživateli

5.4 Další dojmy uživatelů

Aplikace byla zaslána více než deseti uživatelům a mnozí ji velmi detailně otestovali, nahráli různé cesty a nechali se navádět. Obecně byla zpětná vazba velmi pozitivní, aplikace byla stabilní a uživatelé přicházeli s velkým množstvím nápadů pro rozšíření aplikace jako využití i pro děti, kterým nahraje někdo dopředu cestu do školy, sdílení cest a turistické průvodce. Zpětná vazba dala podnět k úpravám popsaným výše a odhalila spoustu míst, kde si uživatel nebyl zcela jistý tím, co má provést a úpravy měly za cíl toto zlepšit. Několik uživatelů si například při nahrávání nebylo jistých, zda se cesta správně nahrává. Bylo proto implementováno zobrazování cesty s nahranými daty už během nahrávání, takže uživateli se ještě neuložená cesta ukazuje na mapě a aktualizuje se po obdržení každé nové GPS souřadnice.

Celkové byla zpětná vazba velmi cenná a ukázala zcela rozdílný přístup vývojáře a uživatele při používání aplikace. Díky této zkušenosti je třeba používání aplikace vždy konzultovat s běžnými uživateli, neboť představa vývojáře o myšlení uživatele může být často velmi mylná.

6 Závěr

V práci jsme nejprve představili platformu Android spolu s dalšími použitými nástroji, které nám pomohly při vývoji. Poté jsme ukázali průběh návrhu mobilní aplikace od diskuze zadání po funkční a grafický návrh. Následně byly vybrané části implementovány za použití mnoha knihoven a technik pro efektivní vývoj a stabilní Android aplikace. Byla diskutována architektura a metodika vývoje Android aplikace, která byla následně využita při vývoji. Aplikace byla publikována a otestována několika uživateli pro získání zpětné vazby a vyladění nedostatků.

Asistence cestování mentálně postižených se ukázala jako mimořádně zajímavý a také obtížný úkol. Pokusili jsme se poskytnout lidem s mentálním postižením a jejich asistentům nový nástroj, který by jim mohl v jejich často nelehké situaci pomoci a věříme, že některým také pomůže. Během práce byl získán nový rozhled o lidech kolem nás a také spoustu nových vývojářských zkušeností, které jistě budou brzy zúročeny v profesním životě.

V současné době se jedná o pokračování vývoje této aplikace, jako součást nového produktu pro pomoc seniorům a aplikace tak bude dále rozvíjena a cílem by mělo být ji v budoucnu nabídnout jako komerční produkt.

V problematice této mobilní aplikace pro asistenci cestování je stále spoustu věcí k vylepšování. Jako návrhy pro vylepšení mohou být přidání více telefonních čísel na asistenta a implementaci automatického přesměrování mezi nimi. Dále implementace zablokování veškeré editace za účelem zamezení nechtěným úpravám uživatelem a umožnit asistentům definovat jejich vlastní texty jako obsah automatickým textových zpráv. Při používání aplikace se stále objevují nové nápady na použití a budou dále konzultovány.

Literatura

[1] FOWLER, Martin. *Inversion of Control Containers and the Dependency Injection pattern* [online]. Martin Fowler, 2004 [cit. 2016-03-28]. Dostupné z: <http://www.martinfowler.com/articles/injection.html>.

[2] FUTURICE OY. *Best practices in Android development* [online]. Futurice Oy, 2016 [cit. 2016-04-10]. Dostupné z: <https://github.com/futurice/android-best-practices>.

[3] GAMMA, E. – HELM, R. – JOHNSON, R. – VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1. vydání. Addison-Wesley Professional, 1994. ISBN 978-0201633610.

[4] GOOGLE, INC. *Android API Guides* [online]. Google, 2016 [cit. 2016-04-10]. Dostupné z: <http://developer.android.com/guide/index.html>

[5] GOOGLE, INC. *Drive API for Android* [online]. Prosinec 2015 [cit. 2016-04-12]. Dostupné z: <https://developers.google.com/drive/android/appfolder>.

[6] GOOGLE, INC. *Material design* [online]. Google, 2015 [cit. 2016-04-02]. Dostupné z: <https://www.google.com/design/spec/material-design/>.

[7] GOOGLE, INC. *Overview of Google Play Services* [online]. Google, 2016 [cit. 2016-04-10]. Dostupné z: <https://developers.google.com/android/guides/overview>.

[8] GRANT, Allen. *Android 4*. Překlad Jakub Mužík. Brno: Computer Press, 2013. ISBN 978-80-2513-782-6.

[9] HARDT, Ed., D. *The OAuth 2.0 Authorization Framework* [online]. Microsoft, 2012 [cit. 2016-04-10]. Dostupné z: <https://tools.ietf.org/html/rfc6749>.

[10] INCLUSION EUROPE. *Zásady úspěšné komunikace s lidmi s mentálním postižením: Doporučené přístupy k lidem s mentálním postižením v rámci jejich celoživotního vzdělávání i mimo něj*. Praha, 2009. ISBN 2-87460-137-3. Dostupné z: <http://www.inclusion-europe.com/pathways2/images/CZ_Recommendations.pdf>.

[11] ISO/IEC. *ISO/IEC 18092* [online]. 2013 [cit. 2016-04-10]. Dostupné z: <https://www.iso.org/obp/ui/#iso:std:56692:en>.

[12] LALANDA, Phillipe. *Shared repository pattern* [online]. Pattern Languages of Programs'98, 1998 [cit. 2016-04-10]. Dostupné z: <http://www.hillside.net/plop/plop98/final_submissions/P24.pdf>.

[13] MAISSAN, Chris. *Curves and Splines* [online]. Maissan Inc. [cit. 2016-04-10]. Dostupné z: <https://maissan.net/articles/simulating-vines/2>.

[14] MOBILE NATIONS. *Android History* [online]. Android Central, 2016 [cit. 2016-04-10]. Dostupné z: <http://www.androidcentral.com/android-history>.

[15] NFC FORUM. *About the Technology* [online]. 2016 [cit. 2016-04-10]. Dostupné z: <http://nfc-forum.org/what-is-nfc/about-the-technology/>.

[16] NFCTECH.CZ. *Web nfctech.cz* [online]. Dostupné z: <http://www.nfctech.cz>.

[17] OLORUNTOBA, Samuel. *S.O.L.I.D: The First 5 Principles of Object Oriented Design* [online]. Scotch.io, 2015 [cit. 2016-04-10]. Dostupné z: <https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design>.

[18] OPEN PCD. *ISO14443* [online]. [cit. 2016-04-10]. Dostupné z: <http://www.openpcd.org/ISO14443>.

[19] SILLARS, Doug. *High Performance Android Apps*. O'Reilly Media, Inc., 2015. ISBN 978-1491913994. Dostupné z: <https://www.safaribooksonline.com/library/view/high-performance-android/9781491913994/>.

[20] SQLITE. *Datatypes In SQLite Version 3* [online]. SQLite [cit. 2016-04-10] Dostupné z: <https://www.sqlite.org/datatype3.html>.

[21] TECHYOURCHANCE. *Model View Controller (MVC) and Model View Presenter (MVP) architectural patterns in Android* [online]. TechYourChance, 2015 [cit. 2016-04-09]. Dostupné z: <http://bit.ly/25RbZ4L>.

A Zdrojové kódy

Kódy celé aplikace a všech dalších zdrojů, včetně zdrojového LaTeX souboru k tomuto dokumentu lze nalézt na přiloženém CD, případně veškeré soubory projektu jsou v aktuální verzi dostupné na adrese https://github.com/jraska/Diploma-Thesis (29.4.2016), kde je verze uložená na CD dostupná pod tagem `thesis-release` v sekci releases.